# Certified Tester
# Automotive Software Tester (CT-AuT)
# Syllabus

v2.1.1

---

International Software Testing Qualifications Board

---


# Copyright Notice

# Revision History

| Version | Date | Remarks |
|---|---|---|
| 2.1.1 | 2025/12/21 | Errata: Fixes to issues identified during German localization of 2.1 by German Testing Board Working Group. |
| 2.1 | 2025/04/30 | Minor Update: Changes to existing content affected by updates of referenced standards, ISTQB glossary and ISTQB CTFL v4. Improved wording for better understandability. Transferred into current syllabus template. See Appendix C – Release Notes for details. |
| 2.0.2 | 2018/07/04 | GA approval of English version (English translation of V2.0 DE) |
| 2.0 DE | 2017/03/31 | Learning objectives and content based on V.1.1.<br>Release of the (corresponding German) edition per GTB WG meeting of 31.03.2017 (Frankfurt a. M.). |
| 1.1 DE | 2015/06/14 | Review of content and comparison to the German ISTQB® Certified Tester Foundation Level syllabus 2011 V1.0.1 and the ISTQB® Glossary V2.2<br>Release per GTB working group meeting of 15.03.2015 (Munich). |
| 1.0 DE | 2011/01/19 | Author: Dr. Hendrik Dettmering, developed upon request of gasq GmbH<br>The copyright was fully transferred to German Testing Board e. V. |

# Table of Contents

# Acknowledgements

This document was formally released by the General Assembly of the ISTQB® in 2018. The revised version 2.1 has been formally released by the CT-AuT Product Owner Horst Pohlmann in accordance with the process.

It was produced by a team from the German Testing Board for the International Software Testing Qualifications Board: Ralf Bongard (lead), Klaudia Dussa-Zieger, Matthias Friedrich, Thorsten Geiselhart, Horst Pohlmann, Ralf Reißing, Alexander Schulz.

The team thanks Michael Humm for technical reviews, and the review team and the Member Boards for their suggestions and input.

The following persons participated in the reviewing, commenting and balloting of this syllabus:

Laura Albert, Ádám Bíró, Darvay Tamás Béla, Yuliia Fomuliaieva, Matthias Hamburg, Jarek Hryszko, Joanna Kazun, Imre Mészáros, Krisztián Miskó, Gary Mogyorodi, Ingvar Nordström, Mirosław Panek, Lukáš Piška, Meile Posthuma, Márton Siska, Klaus Skafte, Radosław Smilgin, Michael Stahl.

The team thanks Gary Mogyorordi for technical editing.

# 0 Introduction

## 0.1 Purpose of this Syllabus

This syllabus forms the basis for the International Software Testing Qualification for the Automotive Software Tester. The ISTQB® provides this syllabus as follows:

1. To member boards, to translate into their local languages and to accredit training providers. Member boards may adapt the syllabus to their particular language needs and modify the references to adapt to their local publications.
2. To certification bodies, to derive examination questions in their local languages adapted to the learning objectives for this syllabus.
3. To training providers, to produce training materials and determine appropriate teaching methods.
4. To certification candidates, to prepare for the certification exam (either as part of a training course or independently).
5. To the international software and system engineering community, to advance the software and system testing profession, and as a basis for books and articles.

## 0.2 The Certified Tester Automotive Software Tester

The Automotive Software Tester qualification is aimed at anyone involved in software testing in the automotive domain. This includes roles such as testers, test analysts, test engineers, test consultants, test managers, user acceptance testers, and software developers. The Automotive Software Tester qualification is also appropriate for anyone who strives for a basic understanding of automotive software testing, such as project managers, safety managers, quality managers, software development managers, business analysts, IT directors and management consultants.

## 0.3 Career Path for Testers

The ISTQB® scheme provides support for testing professionals at all stages of their careers. Individuals who achieve the ISTQB® Certified Tester Automotive Software Tester certification may also be interested in the Core Advanced Levels (Test Analyst, Technical Test Analyst, and Test Manager) and thereafter Expert Level (Test Management or Improving the Test Process). The Specialist stream offers a deep dive into areas that have specific test approaches and test activities e.g. in Agile Testing, AI Testing, or Mobile App Testing, or which cluster testing know-how for certain industry domains e.g. Gambling or Gaming. Please visit www.istqb.org for the latest information on ISTQB's Certified Tester Scheme.

## 0.4 Business Outcomes

This section lists the Business Outcomes expected from a candidate who has achieved the Automotive Software Tester certification.

A person certified as Automotive Software Tester can…

| | |
|---|---|
| AuT-BO1 | Collaborate effectively in a test team. (“collaborate”) |
| AuT-BO2 | Adapt the test techniques known from the ISTQB® Certified Tester Foundation level (CTFL®) to the specific project requirements. (“adapt”) |
| AuT-BO3 | Consider the basic requirements of the relevant standards (i.e., Automotive SPICE® and ISO 26262) for the selection of suitable test techniques. (“select”) |
| AuT-BO4 | Support the test team in the risk-based planning of the test activities and apply known elements of structuring and prioritization. (“support & apply”) |
| AuT-BO5 | Apply the virtual test environments (i.e., MiL, SiL, and HiL) in test environments. (“apply”) |

## 0.5 Learning Objectives and Cognitive Level of Knowledge

Learning Objectives support business outcomes and are used to create the Certified Tester Automotive Software Tester exams.

The specific learning objectives levels are shown at the beginning of each chapter, and classified as follows:

- K1: Remember
- K2: Understand
- K3: Apply

Further details and examples of learning objectives are given in Appendix A.

For all terms listed as keywords just below chapter headings, the correct name and definition from the ISTQB® glossary shall be remembered (K1), even if not explicitly mentioned in the learning objective.

## 0.6 The Certified Tester Automotive Software Tester Certificate Exam

The Certified Tester Automotive Software Tester Certificate exam will be based on this syllabus. Answers to exam questions may require the use of material based on more than one section of this syllabus. All sections of the syllabus are examinable, except for the Introduction, the Appendices, and sections marked as informative only. The exam questions will confirm knowledge of keywords and learning objectives at all K-levels.

Standards and books are included as references, but their content is not examinable, beyond what is summarized in the syllabus itself from such standards and books.

Refer to Exam Structures and Rules document for the Certified Tester Automotive Software Tester Syllabus document for further details.

To take the exam for a Certified Automotive Software Tester candidates must have the ISTQB® Certified Tester – Foundation Level (CTFL®) certificate and an interest in testing in automotive development projects.

However, it is strongly recommended that candidates also:

- have at least a minimum background knowledge in software development or software testing (for example six months experience as a software tester or as a developer), or
- have taken a course, which is accredited per the ISTQB® standard (by an ISTQB®-member-board) and/or
- have gained initial experience in testing Electric/Electronic (E/E) system development projects in the Automotive industry

Entry Requirement Note: The ISTQB® Foundation Level certificate shall be obtained before taking the Automotive Software Tester certification exam

## 0.7 Accreditation

An ISTQB® Member Board may accredit training providers whose course material follows this syllabus. Training providers should obtain accreditation guidelines from the Member Board or body that performs the accreditation. An accredited course is recognized as conforming to this syllabus and is allowed to have an ISTQB® exam as part of the course.

The accreditation guidelines for this syllabus follow the general Accreditation Guidelines published by the Processes Management and Compliance Working Group.

## 0.8 Handling of Standards

International standardization organizations like IEEE and ISO have issued standards associated with quality characteristics and software testing. Such standards are referenced in this syllabus. The purpose of these references is to provide a framework (as in the references to ISO/IEC 25010 regarding quality characteristics) or to provide a source of additional information if desired by the reader. Please note that syllabi are using standard documents as reference. Standard documents are not intended for examination. Refer to chapter '7 References' for more information on Standards.

## 0.9 Level of Detail

The level of detail in this syllabus allows internationally consistent courses and exams. To achieve this goal, the syllabus consists of:

- General instructional objectives describing the intention of the Certified Tester Automotive Software Tester Syllabus
- A list of terms that students must be able to recall
- Learning objectives for each knowledge area, describing the cognitive learning outcome to be achieved
- A description of the key concepts, including references to sources such as accepted literature or standards

The syllabus content is not a description of the entire knowledge area of automotive software testing; it reflects the level of detail to be covered in Certified Tester Automotive Software Tester training courses. It focuses on test concepts and techniques that apply to software projects in the automotive domain.

The syllabus uses the terminology (i.e., the name and meaning) used in software testing and quality assurance according to the ISTQB® Glossary.

For the terminology in related disciplines please refer to the respective glossaries: ISO 26262 for functional safety engineering, and IREB® Glossary for requirements engineering.

## 0.10 How this Syllabus is Organized

There are four chapters with examinable content. The top-level heading for each chapter specifies the time for the chapter; timing is not provided below chapter level. *For accredited training courses, the syllabus requires a minimum of 12* hours of instruction, distributed across the four chapters as follows:

- Chapter 1: Introduction to Automotive Software Tester (30 minutes)
  - The testers learn about the challenges of developing automotive products, including opposing goals, increasing complexity, and the impact of standards on time, cost, quality, and risks.
  - They also learn about the six phases of a product's lifecycle and their role in releasing products.
- Chapter 2: Standards for the testing of E/E systems (300 minutes)
  - The testers learn about ASPICE, covering its capability levels, test-relevant processes, documentation requirements, and the role of a test strategy, along with ASPICE's traceability and test strategy expectations.
  - For ISO 26262, the testers focus on safety culture, automotive safety integrity levels, test techniques, and understand their role within the safety lifecycle.
  - Testers also understand the influence of AUTOSAR on testing and compare the objectives of ASPICE, ISO 26262, and CTFL® and the respective test levels.
- Chapter 3: Testing in a virtual environment (160 minutes)
  - The testers learn about the purpose, structure, and essential functions of automotive virtual test environments, including closed-loop and open-loop systems, and gain an understanding of different XiL test environments (i.e., MiL, SiL, and HiL), their applications, advantages, and limitations.
  - They also learn how to assign appropriate test scopes to each test environment.
- Chapter 4: Automotive-specific static and dynamic test techniques (230 minutes)
  - The testers learn how to explain and apply essential guidelines for code, such as MISRA-C, and requirements, such as ISO/IEC/IEEE 29148.
  - The testers create test cases for modified condition/decision coverage, understand fault injection and back-to-back testing, and select suitable test techniques and test approaches based on project context.

# 1 Introduction to Automotive Software Testing – 30 minutes

**Keywords**

none

**Learning Objectives for Chapter 1**

**1.1 Requirements from Divergent Project Objectives and Increasing Product Complexity**

AuT-1.1. (K2) Explain and give examples of the challenges of automotive product development that arise from divergent project objectives and increasing product complexity

**1.2 Project Aspects Influenced by Standards**

AuT-1.2 (K1) Recall project aspects that are influenced by standards (e.g., time, cost, quality, project risks and product risks)

**1.3 The Six Generic Stages in the System Lifecycle**

AuT-1.3 (K1) Recall the six generic stages in the system lifecycle per ISO/IEC/IEEE 24748-1

**1.4 The Contribution and Participation of the Tester in the Release Process**

AuT-1.4 (K1) Recall the role (i.e., contribution and collaboration) of the tester in the release process

## Introduction

One of the seven principles of software testing is “Testing is context dependent”. This chapter outlines the E/E system development context in which an “Automotive Software Tester”[1] operates. On the one hand, conflicting goals such as cost efficiency, safety requirements, and fast time-to-market, increasing complexity and intense demand for innovative solutions lead to particular challenges. On the other hand, standards and the lifecycle of vehicles form the framework in which the tester is working. In the end, the tester is working to release of software and systems.

## 1.1 Requirements from Divergent Project Objectives and Increasing Product Complexity

Original Equipment Manufacturers (OEMs) and suppliers are launching new car models[2] more frequently than in the past, despite facing increasing cost pressures. The following aspects influence this process:

- Increasing number of models and complexity:
  To be able to better meet individual end-customer needs, OEMs offer more and more car models. However, this reduces the quantities per model. To cover the resulting increases in development and production costs, producers develop several models as varieties of a common platform. The development of a platform, however, is far more complex than the development of a single model because of the need to keep control over the many possible variations.
- Increasing range of functionality:
  The end-customer requests more and more innovations while maintaining existing functions, which causes the range of functions to increase.
- Increasing number of configurations:
  The end-customer wants to adjust their car model to their individual wishes. This requires a wide range of configurations for a single car model, including in terms of functionality.
- Increased quality requirements:
  Despite increasing levels of functionality and complexity, the end-customer expects the same or even higher quality of the vehicle and its functions.

As the project objectives – time, cost, and scope – are competing (i.e., known as the ‘Project management triangle’, with quality often considered an aspect of scope) OEMs and suppliers must strive for a more efficient system development, which allows for shorter development times despite increasing complexity, increasing quality requirements and smaller budgets.

[1] In the following we will only use the term “Tester”: It is to be understood as the short form of “Automotive Software Tester.”

[2] Example from a study by the management consultancy Progenium: “In 1990, only 101 different car models were on offer …, in 2014, this number had increased to 453”
Above this, “the next decade of mobility transformation puts the consumer first and foremost” brings up new challenges

## 1.2 Project Aspects Influenced by Standards

Standards influence major project aspects such as time, cost, quality, project risks and product risks:

- Standards increase the efficiency of processes (e.g., by reducing development time or cost while maintaining stable quality) by:
  - uniform naming
  - better transparency
  - easier collaboration (i.e., internal and external)
  - increased re-usability
  - consolidated experience (‘Best Practice’)
- With well-established technology guidelines, they help to discover risks and defects early and to resolve the identified risks and defects.
- Standards are the basis for audits. Therefore, an auditor can assess the quality of a product or process. At the same time, the auditor can check if the standards meet the requirements.
- Standards are part of the contractual or regulatory provisions and guidelines.

This syllabus references selected standards relevant to automotive software testing, including:

- ISO 26262 and ASPICE, which provides standardized processes and methods
- AUTOSAR, which standardizes software architecture and products

## 1.3 The Six Generic Stages in the System Lifecycle

The system lifecycle of a car and its components[3] begins with the product idea and ends with decommissioning. It involves development, business, logistics, and production technology processes.

Milestones with defined entry criteria and exit criteria ensure mature processes and structure the system lifecycle[4] into six stages with typical test activities[5]:

- Concept: Test planning to define the test strategy and test objectives
- Development: Test analysis, test design, test implementation, test execution, test monitoring, test control, and test completion ensure quality.
- Production: End of line testing to verify readiness
- Utilization: Typically, no test activities are performed.
- Support: Maintenance testing to ensure continued reliability
- Retirement: Migration testing validates decommissioning or data transfer.

The widely used automotive product development process often condenses these into three main stages: concept, development, and production, where most testing and release activities occur.

[3] ECUs (hardware and software) as well as components.
[4] The safety lifecycle of ISO 26262 consists of similar phases.
[5] For test activities see also: fundamental test process.

## 1.4 The Contribution and Participation of the Tester in the Release Process

In the automotive environment, a project reaches a milestone by declaring a release once the goals are verified as achieved. From this point on, the release item is considered to have the maturity required for its intended use. The release item includes the software configuration including parameterization and, if necessary, hardware and mechanics, and the supporting documentation.

The tester delivers important information for the release process via the final test report:

- tested items
- evaluated quality characteristics (e.g., response time and resource usage)
- known defects
- product metrics
- information for release recommendation upon achieving the exit criteria based on the release level as defined in the Best Practice Guidelines (e.g., testing on closed terrain, testing on public roads, and installation recommendations)

Additionally, the tester participates in creating further deliverables relevant for the release:

- prioritize and participate in the decision regarding changes
- prioritize features for the order of implementation

# 2 Standards for the Testing of Electric/Electronic (E/E) Systems – 300 minutes

**Keywords**

functional safety, method table

**Domain Specific Keywords**

automotive safety integrity level (ASIL), software verification, system verification

**Learning Objectives for Chapter 2**

**2.1 Automotive SPICE (ASPICE)**

AuT-2.1.1.1 (K1) Recall the two dimensions of ASPICE

AuT-2.1.1.3 (K2) Explain the capability levels 0 to 3 of ASPICE

AuT-2.1.2.1 (K1) Recall the purpose of the test-specific processes of ASPICE

AuT-2.1.2.2 (K2) Explain the meaning of the four rating levels and the capability indicators of ASPICE from the testing perspective

AuT-2.1.2.3 (K2) Explain the requirements of ASPICE for a test strategy including the criteria for regression verification

AuT-2.1.2.4 (K1) Recall the requirements of ASPICE for testware

AuT-2.1.2.5 (K3) Use verification measures for software unit verification

AuT-2.1.2.6 (K2) Explain the different traceability requirements of ASPICE from the testing perspective

**2.2 ISO 26262**

AuT-2.2.1.1 (K2) Explain the objective of functional safety for E/E systems

AuT-2.2.1.2 (K1) Recall the testers' contribution to the safety culture

AuT-2.2.2 (K2) Discuss the role of the tester in the framework of the safety lifecycle per ISO 26262

AuT-2.2.3.2 (K1) Recall the parts of ISO 26262 that are relevant to the tester

AuT-2.2.4.1 (K1) Recall the criticality levels of ASIL

AuT-2.2.4.2 (K2) Explain the influence of ASIL on test techniques and test types for static testing and dynamic testing and the resulting test scope

AuT-2.2.5 (K3) Use the method tables of ISO 26262

**2.3 AUTOSAR**

AuT-2.3.1 (K1) Recall the project objectives of AUTOSAR

AuT-2.3.3 (K1) Recall the influence of AUTOSAR on the work of the tester

## 2.4 Comparison of ASPICE, ISO 26262, and CTFL®

AuT-2.4.1 (K1) Recall the different objectives of ASPICE and ISO 26262

AuT-2.4.2 (K2) Explain the differences between ASPICE and ISO 26262 and CTFL® regarding test levels

# 2.1 Automotive SPICE (ASPICE)

## Introduction

Process improvement follows the approach that the quality of a system is influenced by the quality of the development process. Process models offer an option for improvement by measuring the process capability of an organizational unit or project compared to the reference model. Furthermore, the model serves as the framework for the improvement of the processes of an organizational unit or project using the assessment results.

Since 2001, the SPICE[6] User Group and the Automotive Special Interest Group (AUTOSIG) developed ASPICE. Since 2005, the standard has been well established in the automotive industry. All statements made in this paragraph refer to ASPICE version 4.0.

## 2.1.1 Design and Structure of the Standard

### 2.1.1.1 The two dimensions of ASPICE

ASPICE defines an assessment model with two key dimensions of assessment:

In the **process dimension**, ASPICE defines the process reference model (PRM). The PRM serves as a reference to compare the organization's processes against it so that they can be assessed and improved. For each process, ASPICE defines the purpose, the results, the required actions (i.e., base practices) and the information items (i.e., work results and work products).

In the **capability dimension,** ASPICE defines several process attributes (i.e., measurable characteristics) that divide the capability level. For each process, there are generic and process specific attributes. ISO/IEC 33020 serves as a basis for the assessment of the process capability.

With the help of this model, it is possible to assess the processes (i.e., process dimension) regarding their capability (i.e., capability dimension).

### 2.1.1.2 Process categories in the process dimension [informative]

ASPICE categorizes processes into groups, which are further classified into categories:

The **primary lifecycle processes** include all processes that serve as key processes:

- Acquisition process group (ACQ)
- Supply process group (SPL)
- System engineering process group (SYS)
- Software engineering process group (SWE)
- Machine learning engineering process group (MLE)
- Hardware engineering process group (HWE)
- Validation process group (VAL)

[6] Acronym for "Software Process Improvement and Capability Determination"

The **supporting lifecycle processes** include all processes that support other processes:

- Supporting process group (SUP)

The **organizational lifecycle processes** include all processes that support the company objectives:

- Management process group (MAN)
- Process Improvement process group (PIM)
- Reuse process group (REU)

For the tester, the process groups system engineering (SYS) and software engineering (SWE) are of special interest, possibly also the MLE, VAL and HWE.

#### 2.1.1.3 Capability levels (CL) in the capability dimension

The assessor assesses the process capability with the help of a six-level assessment system (i.e., display of levels). ASPICE defines the capability levels 0 to 3[7] as follows:

- Level 0 (incomplete process): The process is not implemented or fails to achieve its process purpose. At this level there is little or no evidence of any systematic achievement of the process outcome. The process does not exist or does not achieve the purpose of the process. Example: The tester only checks a small part of the requirements.
- Level 1 (performed process): The implemented process achieves its purpose (but maybe executed inconsistently). Example: There is no complete planning visible for a test process. However, the tester can show the level of fulfilment of the requirements.
- Level 2 (managed process): The project plans and supervises the test process in its test execution. Under certain circumstances, it adapts the course of action during test execution to meet the test objectives. The requirements for the work products are defined. A project member reviews the work products and approves them. Example: The test manager defines the test objectives, plans the test activities and manages the test process. This includes reacting to deviations accordingly.
- Level 3 (established process): The project uses a standardized test process, and findings are used for constant improvement. Example: There is an organizational test strategy. After test completion the test manager works to further develop it.

[7] The capability level 4 and 5 are currently not in the focus of the automotive industry.

## 2.1.2 Requirements of the Standard

### 2.1.2.1 Test-specific processes of ASPICE

ASPICE defines test processes according to all processes of the software and system development[8]:

- Software unit verification (SWE.4) requires static testing and dynamic testing. It assesses the components of the software based on its detailed design (SWE.3).
- Software component verification and integration verification (SWE.5) assess the integrated software based on the software architectural design (SWE.2).
- Software verification (SWE.6) assesses the integrated software based on the software requirements analysis (SWE.1).
- System integration and integration verification (SYS.4) assess the integrated system based on the system architectural design (SYS.3).
- System verification (SYS.5) assesses the integrated system based on the system requirements analysis (SYS.2).

### 2.1.2.2 Rating levels and capability indicators

An assessor can assess the process capability via capability indicators. ASPICE defines them for 9 process attributes (PA). For the capability levels 1 to 3, they are defined as follows (using the example of SWE.6 in parentheses):

- PA 1.1: Process performance process attribute (e.g., the tester follows the test process)
- PA 2.1: Process performance management process attribute (e.g. the tester plans, supervises and controls the test activities)
- PA 2.2: Work product management process attribute[9] (e.g., the tester checks the quality of the testware)
- PA 3.1: Process definition process attribute (e.g., the person responsible for the test process defines an organizational test strategy)
- PA 3.2: Process deployment process attribute (e.g., the tester applies the test strategy defined in PA 3.1)

For the process execution, ASPICE defines two types of capability indicators: base practices (BP) and information items. In addition, generic practices (GP) and information items are defined. The assessment of the process attributes is based on the implementation level of the indicators at four rating levels:

- **N** (None): not fulfilled (0% up to ≤ 15%)
- **P** (Partly): partly fulfilled (> 15% up to ≤ 50%)
- **L** (Largely): largely fulfilled (> 50% up to ≤ 85%)
- **F** (Fully): fully fulfilled (> 85% up to ≤ 100%)

[8] In addition to the test processes mentioned below there is also the process group VAL. The purpose of VAL.1 “is to provide evidence that the end product, allowing direct end user interaction, satisfies the intended use expectations in its operational target environment”.

[9] work product management applies to all documented information

For a process to reach a certain capability level, the capability indicators must be achieved either largely fulfilled (L) or fully fulfilled (F).

#### 2.1.2.3 Test strategy and criteria for regression verification

ASPICE requires a test strategy (i.e., verification measures) for each test process (see section 2.1.2.1) starting with capability level 2. The test manager develops it during the test planning. Test guidelines, project objectives, and contractual and regulatory requirements form the basis for it.

The tester is aware of early testing as a principle of testing. This also applies to the testing of software in the automotive environment. However, another aspect comes into play here because test environments at higher test levels are significantly more expensive. For example, for testing at higher levels, especially developed and embedded hardware is necessary (e.g., as a prototype or unique model). The test strategy defines the level-specific test environments, but also which tests the tester is required to perform in which test environments.

The criteria for regression verification are an essential part of the test strategy. They specify the details for the regression verification[10]. The challenge lies in the economically sensible choice of test cases (i.e., added value of testing). The criteria for regression verification define the test objective and the test approach for the choice of the regression tests. For example, the choice can be risk-based. An impact analysis helps to identify the areas the tester must focus on with regression tests. However, the test manager may also ask the tester to repeat all automated test cases for each release.

#### 2.1.2.4 Testware in ASPICE

ASPICE 4.0 does not impose requirements on testware. Instead, it establishes requirements concerning the documentation of specific information that arises during testing. For test activities, ASPICE requires the following information items (II):

- 08-60: Verification measures (e.g., simulations, dynamic tests and static tests), typically documented in a test plan
- 15-52: Verification results (e.g., documented verification logs, test reports and defect reports)

For each information item, ASPICE defines examples of information item characteristics (IIC) and content. They serve as an objective indicator for process execution.

ISO/IEC/IEEE 29119-3 can be used to further structure information items.

#### 2.1.2.5 Verification measures for software unit verification

The tester verifies compliance with the software detailed design and with the functional and non-functional requirements according to the verification measures. The measures define how the tester provides the evidence. The tester can use combinations of static and dynamic test techniques to verify the software units.

[10] similar to the regression-averse test strategy in ISTQB

In SWE.4.BP.1, ASPICE requires the definition of verification measures (see section 2.1.2.4) for the verification of software units. This allows the tester to assess the extent to which the software unit meets its requirements. The following examples could be used as software unit verification measures:

- Software unit test cases including test data
- Tool-supported static analysis, which assesses compliance with coding guidelines (e.g., MISRA-C, see section 4.1.1)
- Reviews for software units or parts of software units, which cannot be assessed by tool-supported static analysis

If a developer changes a software unit, the tester must also evaluate this change. Therefore, the verification measures of software units also include the criteria for regression verification. This includes the verification of the changed code, the confirmation testing, and the repeated verification of the non-changed parts (static and dynamic regression tests).

#### 2.1.2.6 Traceability in ASPICE

As in the CTFL® Core Syllabus [ISTQB_FL_SYL], ASPICE additionally requires bidirectional traceability[11]. This allows the testers to:

- Analyze impact, e.g., change impact analysis
- Evaluate coverage
- Track status

Moreover, this allows the tester(s) to ensure the consistency between the linked elements, textually and semantically.

ASPICE differentiates between vertical and horizontal traceability:

**Vertically:** ASPICE requires that stakeholder requirements be linked across all levels to the software units. In doing so, the link over all levels of development ensures consistency between the related work products.

**Horizontally:** ASPICE requires traceability and consistency, between the work results of the development and the corresponding test specifications and test results.

In addition, the base practice SUP.10.BP4 requires bidirectional traceability between change requests and work products affected by the change requests. Because change requests are often initiated by a defect, bidirectional traceability is established between change requests and the corresponding defect reports. Because of the occasionally large number of links, a consistent chain of tools can be helpful. This allows the tester to efficiently create and manage the dependencies.

## 2.2 ISO 26262

### 2.2.1 Functional Safety and Safety Culture

#### 2.2.1.1 Objective of functional safety for E/E systems

The functional and technical complexity of embedded systems is constantly rising. At the same time, powerful software-based electrical and electronic systems allow new complex functionalities such as the automation of driving functions in a car.

[11] In the following, the term traceability will always imply the bidirectional traceability.

Due to the increasing complexity, the risk of an erroneous action happening during development is increasing as well. The consequence can be an unrecognized defect in the system. For systems with an inherent risk potential for life and limb, the responsible safety engineer or safety manager therefore needs to analyze potential risks. If there is an actual risk, suitable measures need to be identified to mitigate the possible risk impact to an acceptable risk level.

The methods for the execution of such analyses are summarized in the standards for functional safety. The foundation standard is IEC 61508. The International Organization for Standardization (ISO) adapted ISO 26262 from this standard, which is available in its second edition from 2018.

Functional safety involves ensuring that E/E systems operate without causing unreasonable risk due to hazards from malfunctioning behavior. In this sense, the term is to be differentiated from other related terms such as information security (or cybersecurity), product safety or occupational safety.

Occupational safety and cybersecurity are not in the focus of ISO 26262. However, a lack of cybersecurity can endanger functional safety. Both functional safety and cybersecurity contribute to product safety.

#### 2.2.1.2 Contribution of the tester to the safety culture

Within the product development according to ISO 26262, monitoring only your own organization's processes is not sufficient. All participants need to follow a cross-process approach. Everybody must understand their impact on the development process and the safety of the final product. This includes external partners and suppliers.

The participants must understand that their own actions do not happen independently of other processes. Each step of development constitutes an essential contribution to both compliance with and implementation of the safety-relevant requirements. This responsibility does not end with the market launch. It continues until the end of the safety lifecycle.

The testers contribute to the safety culture by participating responsibly in the software development lifecycle phases and by carrying out their work with a continuous view of the overall context of the product development.

### 2.2.2 Integration of the tester in the safety lifecycle

The safety lifecycle describes safety-oriented activities during product development. It starts with the first product idea and the identification of possible risks. After the specification of resulting safety requirements, the implementation into a specific product follows. The safety lifecycle ends with the disposal of the product at the end of its life.

The safety lifecycle according to ISO 26262 goes through the following phases:

- 1st phase: Concept phase
- 2nd phase: Product development. This phase ends with the “release for production”.
- 3rd phase: Production, operation, service and decommissioning

The tester works primarily in the first two phases. Changes to the product within the third phase led to a return to the first or second phase, depending on their extent. Therefore, the tester also participates in modifications. Based on the safety-related requirements, test techniques are selected, and test cases are designed for the verification and validation of these requirements. The tester will then perform those tasks in the respective subphases of the product development.

Test planning normally takes place within the concept phase. Adjustments to the resulting documents (e.g., in the test plan or the test specifications) can, however, be necessary in any phase. Test execution takes place primarily at the transitions between the individual subphases of product development. For example, transitioning from implementation to software integration, and then to hardware-software integration. Moreover, the testers significantly contribute to the transition to the third phase with their test activities.

## 2.2.3 Structure and Testing of Specific Parts of the Standard

### 2.2.3.1 Design and structure of the standard [informative]

ISO 26262 consists of 12 parts:

- Vocabulary (part 1)
- Management of functional safety (part 2)
- The phases of the safety lifecycle:
  - Concept phase (part 3)
  - Product development at the system-, hardware- and software level (parts 4-6)
  - Production, operation, service and decommissioning (part 7)
- Supporting processes (part 8)
- ASIL-oriented and safety-oriented analysis (part 9)
- Guidelines on ISO 26262 (part 10)
- Guidelines on application of ISO 26262 to semiconductors (part 11)
- Adaptation of ISO 26262 for motorcycles (part 12)

Apart from part 1, part 10 and part 11, each part contains:

- A general introduction
- The scope of application
- Normative references
- Requirements for compliance with the standard

These are followed by the specific topics of the corresponding part. The structure of their description is the same in each part. The activities that are to be carried out are described via a recurring structure in all parts:

- Objective

- General information
- Introductory information
- Prerequisites
- Further supporting information
- Requirements and recommendations
- Work results

#### 2.2.3.2 Relevant parts of ISO 26262 for the tester

For the software tester, the software verification and, at least partly, the system validation, are paramount. Apart from part 1 (vocabulary), several other parts are also of special interest: parts 4 and 6 provide detailed information and requirements regarding recommended software verification measures. This applies to the selection, design, implementation, and execution of the corresponding verification measures.

In doing so, these parts focus on the test and verification-specific aspects of the system level (i.e., part 4, including safety validation) and the software level (i.e., part 6). If hardware-specific aspects are also relevant for this work, the tester will find those in part 5. Aspects concerning both hardware and software are considered within the scope of the hardware software interface (parts 4-6).

Part 8 has a special role as it describes the process-specific characteristics of verification at all test levels. In addition, it contains requirements for supporting processes that are critical for ensuring standardized processes, such as configuration management and the qualification of tools.

In part 12, the tester will find adapted method tables based on the newly introduced Motorcycle Safety Integration Level (MSIL) assessment for motorcycle projects.

### 2.2.4 The Influence of Criticality on the Extent of the Test

#### 2.2.4.1 The criticality levels of ASIL

The ASIL is a measure for the required risk reduction through measures of functional safety. Such measures can be, for example, an independent safety function for monitoring an E/E system or the implementation of specifically defined methods. For higher risk levels, more elaborate measures can be necessary.

At the beginning of the project, a team of experts performs a hazard analysis and risk assessment (HARA) for the product. For each hazard identified by this analysis, the team determines an ASIL using the methodologies defined in the standard. In the next step, the team defines safety goals and safety requirements. These have the same ASIL as the underlying hazard.

ISO 26262 defines four levels: from ASIL A for low, up to ASIL D for high safety requirements.

If the results of the hazard analysis and risk assessment lead to requirements below ASIL A, then, in terms of the standard, those are not safety relevant. For such requirements, there are not any functional safety-related actions needed, and it is enough to cover such requirements using already existing quality management systems (QMS). For practical reasons - those requirements are often marked "QM" - to distinguish them from ASIL-related ones.

#### 2.2.4.2 Influence of ASIL on test techniques, test types and the resulting test scope

The determined ASIL directly influences the extent of the tests to be carried out by the tester. Depending on the level of the ASIL, ISO 26262 recommends the execution of different measures or packages of measures. In doing so, the standard recommends more extensive and detailed measures for higher ASIL. For lower ASIL levels, the execution of the specified measures is often optional.

ISO 26262 specifies three levels of recommendations: no recommendation, recommended, and highly recommended. The general understanding of these recommendation levels is that where a method is listed as “highly recommended”, it must be applied, while a method listed as “recommended” should be applied or a rationale for not using it shall be given. In the case of “no recommendation”, the standard does not provide any recommendation for or against the use of the measure in question. It can be used as a supportive measure without any concern. However, its execution does not replace the measures recommended or highly recommended by ISO 26262.

For the tester, this means that the standard recommends specific test techniques and test types for safety-relevant systems depending on the ASIL. The tester can only decide freely to the extent that the standard allows for the special case. For example, the use of equivalence partitioning and boundary value analysis are merely recommended for ASIL A. On the other hand, for an ASIL B or higher, those techniques are highly recommended (see section 2.2.5).

The ASIL is not a characteristic of the entire product. It is tied to a specific safety goal and safety requirements derived from it. For the same product, safety requirements with different ASILs can therefore result in significantly different test efforts. This must be taken into consideration by the tester when planning the test scope.

### 2.2.5 Application of Content from the CTFL® in the Context of ISO 26262

ISO 26262 offers the tester specific recommendations for test activities in the form of method tables. These can be found in parts 4, 5, 6, 8 and 12. Apart from functional safety specific recommendations for test processes and test activities, they also include the test techniques to be used.

In this context, the standard uses the term “method” related to all applicable test techniques or test activities. At this point, the functional safety terminology differs slightly from the terms of the ISTQB®. For the tester, the following methods of ISO 26262 are of special interest:

- Test techniques (e.g., equivalence partitioning and boundary value analysis)
- Techniques for test execution (e.g., simulation or prototype of a component or system)
- Test types (e.g., non-functional tests such as performance testing and endurance testing)
- Test environments (e.g., hardware-in-the-loop and vehicles)
- Static test techniques (e.g., reviews and static analysis)

The method tables define the methods recommended by the standard for each ASIL level.

They are always designed in the same structure:

| | ASIL A | ASIL B | ASIL C | ASIL D |
|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| 1 | Method x | o | + | ++ | ++ |
| 2 | Method y | o | o | + | + |
| 3a | Method z1 | + | ++ | ++ | ++ |
| 3b | Method z2 | ++ | + | o | o |

Table 1: Example of a method table

For each method, depending on the ASIL level, it is documented whether its use is recommended (+) or even highly recommended (++). For methods marks as optional (o), there is no recommendation provided by the standard for or against its use.

ISO 26262 also lists equivalent alternative methods in the method tables using letter suffixes (in the example above, rows 3a and 3b). Here, the tester needs to choose a suitable combination to be able to verify the relevant requirements in an ASIL-compliant way. The tester must justify the usefulness of the selected combination.

In the case of methods without alternatives (e.g., rows 1 and 2), this option of choice is not permitted. Here, the tester must apply all methods that are highly recommended for the respective ASIL level.

For example, when verifying a requirement according to ASIL C, the following methods are derived from table 1:

- Method x: highly recommended, so it must be applied when developing in accordance with ISO 26262
- Method y: recommended, so it should be applied or a rationale for not applying the method has to be given
- Methods z1 and z2: here, at least method z1 should be chosen as it has the highest level of recommendation for ASIL C

ISO 26262 also allows the tester to use other methods than the ones listed in the method tables. In this case, the method chosen must be justified for its usefulness and suitability.

Generally, ISO 26262 allows to customize the safety activities to fit the specific needs of a particular project ("tailoring"). For test-related activities, this means that the test strategy and test approach can be tailored to the specific project.

## 2.3 AUTOSAR

AUTOSAR stands for 'AUTomotive Open System Architecture'. AUTOSAR is both the architecture and the development partnership behind it. The AUTOSAR partnership was established in 2003 and includes mainly OEMs and suppliers from the automotive industry. The goal of the partnership is to jointly develop and establish an open industry standard for automotive software architecture. Today, AUTOSAR is a standard established globally for automotive E/E systems. Therefore, the tester certainly encounters AUTOSAR work products. For the tester, it is important to know the project objectives of AUTOSAR, the general structure of the AUTOSAR architecture, and their influence on the work of the tester.

### 2.3.1 Project Objectives of AUTOSAR

The following are the project objectives of AUTOSAR:

- Support the portability of software
- Support the scalability across different architectures and hardware variants
- Support different functional domains
- Support data exchange with non-AUTOSAR systems
- Define an open architecture for automotive software
- Support the development of dependable systems characterized by availability, reliability, safety, integrity, maintainability and usability
- Support the collaboration between partners
- Support applicable international automotive standards and state-of-the-art technologies

### 2.3.2 General Structure of AUTOSAR [Informative]

There are two architecture variants in AUTOSAR: Classic and Adaptive. For simplicity, this syllabus focuses on AUTOSAR Classic. An AUTOSAR system typically consists of several electronic control units (ECUs). On each ECU, the software architecture of AUTOSAR Classic consists of three layers:

- The top application layer is hardware independent. A component of this layer is called AUTOSAR software component (SW-C).
- The bottom hardware-oriented layer is the standardized basic software (BSW).
- The intermediate abstraction layer is the AUTOSAR runtime environment (RTE). As a kind of middleware, it implements the data flow between SW-Cs as well as between SW-Cs and BSW.

In the AUTOSAR methodology for developing automotive systems, OEMs and suppliers exchange system information using description files based on AUTOSAR templates (so-called "arxml-files"):

- The "ECU configuration description" includes data for the integration of the SW-Cs on a single ECU.
- The "system configuration description" includes data for the integration of all ECUs in a vehicle.
- The "ECU extract" includes the data from the "system configuration description" for a single ECU.

### 2.3.3 Influence of AUTOSAR on the Work of the Tester

AUTOSAR influences the work of the tester, especially at the following test levels[13]:

- Software component testing and software integration testing in a virtual environment (e.g. software-in-the-loop): Using a simulation of the BSW and the RTE, the tester can test a SW-C early.
- Software component testing and software integration testing in the real ECU: Here, the tester gets access to the communication on the RTE. Thus, the tester can observe and control the behavior of a SW-C at runtime.
- The AUTOSAR acceptance test is a test of the software system which ensures the compliance of the AUTOSAR functionality at the communication and application levels. The execution of the AUTOSAR acceptance test is optional.
- System integration testing: integration of various ECUs, for example, to test functionality whose implementation is distributed over various ECUs. By simulating not yet implemented functionality, the tester can assess the system behavior early.

## 2.4 Comparison of ASPICE, ISO 26262, and CTFL®

### 2.4.1 Objectives of ASPICE and ISO 26262

There are several standards that propose requirements for product development. Typically, these highlight different aspects in the development. In this subsection ISO 26262 and ASPICE are compared regarding their objectives.

ISO 26262 has the objective of avoiding risks from failures in both hardware and software by providing suitable requirements and processes. For the development of E/E systems, it defines the requirements for the test processes and methods[14] to be used by the tester. These depend on the ASIL level of the item.

ASPICE serves the purpose of determining the capability of the product development process within the framework of assessments. To do so, ASPICE defines assessable criteria for these processes. In contrast to ISO 26262, these are independent of the product's ASIL level.

### 2.4.2 Comparison of Test Levels between ASPICE, ISO 26262, and CTFL®

Both ISO 26262 and ASPICE describe test levels. However, these are not fully consistent with the test levels from CTFL®. Therefore, for an efficient and effective collaboration within the development team, testers should have a common understanding of all test levels.

The term "system" used in ASPICE, and the terms "system" and "item" used in ISO 26262 refer to a product consisting of hardware and software components. The CTFL-Syllabus [ISTQB_FL_SYL], however, refers to software only when using the term "system". Therefore, the test levels of ISTQB® can be mapped to the test levels in ISO 26262 and ASPICE as follows. Note that in the ISO 26262 column the numbers in parenthesis refer to a specific part of ISO 26262 and to a particular chapter therein. In the ASPICE column the parenthesis shows the respective process ID.

| ISTQB CTFL® | ISO 26262:2018 | ASPICE 4.0 |
|---|---|---|

[13] for an overview of test levels see section 2.4.2
[14] for methods in ISO 26262, see section 2.2.5

| | | |
|---|---|---|
| **Acceptance testing** | Safety validation (4-8)[15] | Validation (VAL.1)[16] |
| **System of systems testing**[17] | System and item integration and testing (4-7)[18] | System verification (SYS.5) |
| **System integration testing** | | System integration and integration verification (SYS.4) |
| **System testing** | Testing of the embedded software (6-11)[19] | Software verification (SWE.6) |
| **Component integration testing**[20] | Software integration and verification (6-10) | Software component verification and integration verification (SWE.5) |
| **Component testing**[21] | Software unit verification (6-9) | Software unit verification (SWE.4) |

Table 2: Assignment of the test levels

In the ISTQB CTFL® Syllabus [ISTQB_FL_SYL], most test techniques can be used at different test levels. Similarly, ASPICE does not generally assign any test techniques to test levels. Therefore, both leave the choice to the testers. In ISO 26262, there are individual method tables for each test level (see sections 2.2.4 and 2.2.5). These provide the tester with recommendations of test techniques based on the ASIL level.

[15] The safety validation only covers parts of an acceptance test per ISTQB.

[16] Validation (VAL.1) only covers parts of an acceptance test per ISTQB.

[17] The testing of several heterogenic distributed systems

[18] Item integration and test includes three phases: the integration and the testing of hardware and software of an element, the integration and the testing of all elements belonging to the item, and the integration and the testing of the item in connection with other items in the vehicle.

[19] Test of functional safety software (system) requirements executed on target hardware

[20] The processes of ASPICE (SWE.5) and ISO 26262 (6-10) additionally cover component testing (in the sense of “component”) in CTFL.

[21] The processes of ASPICE (SWE.4) and ISO 26262 (6-9) cover component testing (in the sense of “unit”) in CTFL.

# 3 Testing in a Virtual Environment – 160 minutes

**Keywords**

environment model, hardware-in-the-loop, model-in-the-loop, software-in-the-loop

**Domain Specific Keywords**

closed-loop system, open-loop system

**Learning Objectives for Chapter 3**

**3.1 Test Environment in General**

AuT-3.1.1 (K1) Recall the motivation behind a test environment in automotive development

AuT-3.1.2 (K1) Recall the general parts of an automotive specific test environment

AuT-3.1.3 (K2) Recall the differences between closed-loop systems and open-loop systems

AuT-3.1.4 (K1) Recall the essential functions, databases and protocols of an ECU

**3.2 Testing in XiL Test Environments**

AuT-3.2.1.1 (K1) Recall the structure of a MiL test environment

AuT-3.2.1.2 (K2) Explain the application area and the boundary conditions of a MiL test environment

AuT-3.2.2.1 (K1) Recall the structure of a SiL test environment

AuT-3.2.2.2 (K1) Recall the application areas and the boundary conditions of an SiL test environment

AuT-3.2.3.1 (K1) Recall the structure of a HiL test environment

AuT-3.2.3.2 (K2) Explain the application areas and the boundary conditions of a HiL test environment

AuT-3.2.4.1 (K2) Summarize the advantages and disadvantages of testing using criteria for the XiL test environments

AuT-3.2.4.2 (K3) Apply criteria for the assignment of a given extent of the test to one or more test environments

AuT-3.2.4.3 (K1) Outline the XiL test environments in the V-model

# 3.1 Test Environment in General

## 3.1.1 Motivation for a Test Environment in Automotive Development

The tester faces special challenges. On one hand, the tester is expected to start testing as early as possible to find defects early in the development process. On the other hand, the tester needs a realistic environment to test the system and to find the defects that would appear in the completed product. The tester can solve this conflict by using suitable test environments that match the different development phases. In doing so, the tester can implement and execute individual test tasks before the completely produced or developed ECU is available. By using various test environments, the tester can simulate situations that are hard to reproduce in actual vehicles, such as short circuits and network communication overloads, short circuits and open circuits in wiring harnesses, or overloads in network communications.

## 3.1.2 General Parts of a Test Environment

For the tester to be able to perform testing, they need a test environment in which the missing parts are simulated. This environment helps the tester to control the inputs of the test object and to observe the test results, also called ‘point of control’ (PoC) and ‘point of observation’ (PoO). According to ISO/IEC/IEEE 29119-1, a test environment consists of the following parts:

- Hardware of the test environment (e.g., a control computer, a real-time capable computer when needed, test bench, and development kit)
- Software of the test environment (e.g., operating system, simulation software, and environment models)
- Facilities of communication (e.g., network access and a data logger)
- Tools (e.g., oscilloscope and measuring tools)
- Laboratory (e.g., protection from electromagnetic radiation and noise)

An important part of the test environment is the environment model. Models are key elements of the virtual test environment. They represent aspects of the real world such as the combustion engine, transmissions, vehicle sensors and ECUs or even the driver and road conditions. The test environment also provides different access points. These allow the tester to observe and measure the test object, and to influence or control it when required.

## 3.1.3 Differences Between Closed-loop and Open-loop Systems

The test environment is used to control the input interfaces of the device under test and monitor its output through the output interfaces. Afterwards, the behavior at the output interfaces is analyzed. A success occurs when the actual results match the expected results.

Generally, there are two types of control systems, closed-loop and open-loop. The difference relies on the way the ECU reacts to its environment, and this requires different simulation requirements for the virtual test environment.

#### 3.1.3.1 Closed-loop System

The stimulation in a closed-loop system, also known as in-the-loop, takes the output of the test object into consideration. This is done via an environment model, which collects the outputs and forwards them directly or indirectly to the input of the test object. Therefore, a control loop is created in the test environment.

Closed-loop systems are used more often to test controllers. By using this, complex functions can be tested such as motor and gear controls and driver assist systems such as the anti-lock braking system (ABS®) or the vehicle dynamics control (VDC®).

#### 3.1.3.2 Open-loop System

In an open-loop system, the outputs of the system have no relation to the inputs. The system does not have a feedback loop. In this case, the inputs of the test object are directly defined by the tester in the test procedure.

The application case for open-loop and closed-loop systems depends strongly on the test object behavior. Closed-loop systems have outputs that influence inputs, creating feedback, whereas open-loop systems lack this feedback mechanism. If the test object has a reactive behavior or if it mirrors a state machine, an open-loop system is preferred. In the interior and chassis electronics there are many examples of open-loop systems (e.g., lights and switches).

### 3.1.4 Databases and Communication Protocols of an ECU

An ECU in the automotive environment functions as an embedded system, which consists of hardware and software. The ECU receives different analogue and digital inputs, which constantly collects environmental data in the form of voltage, current and temperature. Moreover, communication bus systems offer additional data from either sensors or other control units. Communication bus systems provide further information to the control unit, coming from sensors or other ECUs, which either collect and process the information themselves or generate them. The test object manages the data in the memory to process the output action, information or data. The generated outputs are also carried out via analogue and digital output pins, bus systems or diagnostic interfaces.

The databases are data warehouses and define the input and output signals of the control unit. These data also include descriptions, control units and conversion formulas of the signals.

The communication protocols describe the data exchange via the corresponding physical interfaces. These protocols define which voltage or bit sequence represents which value of the signal.

The selection of databases and communication protocols depends on the ECU's function. For example, to access diagnostic functions in the control unit, the tester needs the information about the used database (e.g., application programming interface specification (Association for Standardization of Automation and Measuring Systems (ASAM) MCD-3 D) and service-oriented vehicle diagnostics (ASAM SOVD)) and the communication protocol (e.g., unified diagnostic services per ISO 14229, and AUTOSAR specification (SOME/IP)). Other automotive-specific databases are defined in the ASAM and AUTOSAR standards.

## 3.2 Testing in XiL Test Environments

In the automotive industry, the following types of XiL[22] test environments are used:

- Model-in-the-loop (MiL)

[22] The letter X in XiL is a placeholder for the specific test environments in the given list.

- Software-in-the-loop (SiL)
- Processor-in-the-loop[23] (PiL)
- Hardware-in-the-loop (HiL)
- Vehicle-in-the-loop[24] (ViL)

The tester should become familiar with the test environments (i.e., MiL, SiL and HiL) and understand them. The following paragraphs look deeper into the structure and the application areas of the different test environments. XiL in this sense stands as a generic term for the different test environments.

## 3.2.1 Model-in-the-loop (MiL)

### 3.2.1.1 Structure of a MiL test environment

In a MiL test environment, the test object is available as a model. This model is executable but not compiled for special hardware. Developers create models using special modelling tools. The tester needs a test environment to be able to execute and test those models. This is usually implemented in the same development environment as the test object itself. This test environment can additionally contain an environment model. The tester can control and observe the test object via access points. The access points can be placed arbitrarily in the model of the test object and in the environment model. The model of the test object is connected to the environment model and can easily be implemented and used as a closed-loop system.

### 3.2.1.2 Application areas and boundary conditions of a MiL test environment

With a MiL test environment, the tester can test the functional system design. During the development (e.g., the general V-model) the tester can also test single components up to an entire system. To execute the test, the tester needs a computer and the corresponding simulation software including the environment model. The environment model becomes more complex as the scope of functions of the test object increases. The aspects of reality and environmental factors are very complex. The execution times for the models also increase disproportionately. Therefore, the effort to implement a MiL test environment becomes no longer worthwhile from a certain phase of the development.[25]

[23] This test environment is not considered in this syllabus and is purely informative.
[24] This test environment is not considered in this syllabus and is purely informative.
[25] This is also valid for all other XiL environments.

By using a MiL test environment, the tester can test the functional suitability of models over all development levels at an early phase of development (i.e., the left side of the V-Model[26]). But it is not common to enable the environment model to simulate bus or diagnostic functions or physical behavior such as cable breaks or shorts. These tasks can be carried out more easily and at less cost with other test environments.

In a MiL test environment, test execution does not take place in real time. As all components are available as a model, the test execution runs in simulation time. The more complex a system, the more execution time or power the computer needs to provide all necessary information. The duration of the simulation in smaller systems is shorter than the execution in real time. However, a big advantage is that the tester can pause the simulation at any time for detailed analysis and evaluation.

## 3.2.2 Software-in-the-loop (SiL)

### 3.2.2.1 Structure of a SiL test environment

The test object is compiled for a specific SiL test environment. This means the source code has been compiled for a certain computer architecture. This machine code is readable by the test environment as it consists of binary data sets. For the test environment to be able to access signals, a wrapper[27] is necessary. A wrapper is additional software that provides access to inputs that might not be accessible if the test object communicates with the environment directly and not through the wrapper. Therefore, the tester can control software signals and observe them. The wrapper defines the access points to the test object but does not perform its functional tasks.

An environment model is needed for the simulation. The test object is connected to the test environment with the help of the wrapper. The test execution is carried out on a computer without special hardware. The tester needs a software tool that can create a wrapper for the test object with access points to the test environment.

### 3.2.2.2 Application areas and boundary conditions of a SiL test environment

If the developer generates source code based on a model, the real behavior of the software can be different than the expected behavior. This can be caused by different data types in the model, mostly floating point, and in the compiled software code, mostly fixed point, but also by different memory spaces. These anomalies in the expected behavior can be tested for the first time in a SiL test environment. The tester can use a test approach like back-to-back testing (see section 4.2.2) to compare the behavior.

The tester runs the tests, analogous to the MiL test environment, in simulation time. Depending on the calculation technique and the complexity of the environment model, this simulation time can be shorter or longer than in real time. The tester can pause the test execution at any time for detailed analysis and evaluation. Functional suitability tests, interface tests and regression tests are very common test types that can be evaluated in a SiL test environment. On the other hand, performance efficiency tests and reliability tests are unusual. These quality characteristics are mostly affected by the target hardware.

[26] Further descriptions of the content of the V-model can be found at: Sommerville, I. (2011). Software Engineering (9. Auflage). Pearson Education.

[27] Detailed information on software architectures can be found at: Balzert, H., Schröder, K., & Kern, U. (2022). Grundlagen der Softwaretechnik (6. Aufl.). Carl Hanser Verlag.

## 3.2.3 Hardware-in-the-loop (HiL)

### 3.2.3.1 Structure of a HiL test environment

If the test object is available as a prototype or if it is already completely developed, the tester can use a HiL test environment to execute tests. The typical parts of a HiL test environment are:

- A power supply to set different supply voltages
- A real-time capable computer for the environment model to run on
- Several real parts that are not implemented in the environment model
- A signal processing unit of signal type and signal amplitude
- A fault insertion unit (FIU) (see section 4.2.3) for the simulation of cable breaks and shorts
- A breakout box as an additional access interface in the cable harness
- A remaining bus to simulate the non-existent bus participants

### 3.2.3.2 Application areas and boundary conditions of a HiL test environment

The access points in a HiL test environment are diverse. The tester must be aware that using the wrong access points to the test object can render the test results useless. Knowing the different access points and their connections in the HiL test environment enables effective tests to be implemented, executed and evaluated.

The HiL test environment is more complex than the previously mentioned test environments (i.e., MiL and SiL) due to its several parts. The tester must master this complexity to address test tasks. The HiL test environment can be used for component tests, integration tests and system tests. The objective is, among other things, to find functional and non-functional defects in the software and hardware.

With the help of HiL test environments, different test levels can be analyzed. If the test object is a single ECU, it is called a component[28] HiL. If the test object is a combination of several ECUs, it is called a system HiL. The tester uses the component HiL to test functions of the control unit. In the system HiL, the focus is on testing the data exchange between the ECUs and the system test.

In contrast to the previously mentioned test environments (i.e., MiL and SiL) the simulation time in a HiL test environment always runs in real time. The reason for this is that the software is running on real hardware. Pausing or stopping is no longer possible in this test environment. Therefore, the test environment includes a real time capable computer that can collect and serve all relevant signals within a predetermined period of time.

## 3.2.4 Comparison of the XiL Test Environments

### 3.2.4.1 Advantages and disadvantages of testing in the XiL test environments

The tester understands the attributes of the different test environments. In doing so, the tester can understand and assess the advantages and disadvantages of testing in each environment. The criteria are shown in table 3.

[28] The term component is in this case used for an ECU in the context of a E/Esystem.

| Criteria | HiL Test Environment | Impact | MiL | SiL | HiL |
|---|---|---|---|---|---|
| Closeness to reality | Reality is simulated, many characteristics are abstracted, and the focus is on the structures and logic. | Low | + | o | o |
| | Compiled real software can be executed without hardware. | Low to medium | o | + | o |
| | Integrated system, able to run | High | o | o | + |
| Time and effort in debugging | Defects are found in the model of the test object (model adjustment). | Low | + | o | o |
| | Defects are found in the programmed software (software adjustment). | Medium | o | + | o |
| | Defects are found in the system level (system adjustment). | High | o | o | + |
| Effort for implementation and maintenance | Create environment model | Low | + | o | o |
| | Create environment model and wrapper | Medium | o | + | o |
| | Create environment model and wire the hardware components | High | o | o | + |
| Effort for test preparation | Environment can set up quickly. | Low | + | o | o |
| | Environment can set up quickly. | Medium | o | + | o |
| | Design, implementation and evaluation of the tests require high effort. | High | o | o | + |
| Necessary level of maturity of the test object | System models are simulated. | Low | + | o | o |
| | Initial functions are tested with the target software. | Medium | o | + | o |
| | One or more executable ECUs or partial systems are tested as entirely as possible | High | o | o | + |
| Necessary level of detail of the test basis (specification) | Models are tested that partially cover an incomplete specification. | Medium | + | o | o |
| | The relevant information on the software level must be available (e.g., detailed component specification). | Medium to high | o | + | o |
| | Requirements can be tested on the system level with a complete system specification | High | o | o | + |
| Access to the test object | All signals in a model can be observed and controlled. | High | + | o | o |

| Criteria | HiL Test Environment | Impact | MiL | SiL | HiL |
|---|---|---|---|---|---|
| | Only the signals available in the wrapper can be observed and controlled. | Medium | o | + | o |
| | Only the signals available in the hardware or communication protocols can be observed and controlled. | Low | o | o | + |

Table 3: Criteria and their impact on MiL, SiL and HiL test environments

#### 3.2.4.2 Apply criteria for the assignment of a given extent of the test to one or more test environments

In the following table test objectives are described in detail and assigned to suitable test environments.

| Test objective | Description by Examples | MiL | SiL | HiL |
|---|---|---|---|---|
| Test customer requirements | Correct provision of the required functionality. This includes the correct processing of inputs, the correct reaction to inputs and the correct data output at the exit point. | O | O | + |
| Test mechanisms for defect detection and handling | • Detection and handling of random hardware defects<br>• Detection and handling of software defects<br>• Transfer to a safe state after defects are detected, e.g., deactivation of a system | + | + | + |
| Test reaction to configuration data | Check the influence of configuration data on the behavior of the test object | O | + | + |
| Test diagnostic functions | Correct provisioning of the required diagnostic functionality, such as defect detection, defect activation and reset requirement, and defect activation in the defect memory (e.g., on-board diagnostics or in the garage) | - | + | + |
| Test interaction at interfaces | Check internal and external interfaces of the test object | O | + | + |
| Prove usability | The test object should meet the user's usability requirements. | - | O | + |
| Key: + recommended, o possible, - not sensible | | | | |

Table 4: Comparison of test types in MiL, SiL and HiL test environments

This table shows that test environments can be suitable for certain test objectives. This diversified approach becomes evident especially in the testing of the mechanisms for defect detection and handling. In accordance with shift left the general conclusion is that basic requirement and design defects are already detected early through testing. Therefore, MiL is used for detection of general design defects, SiL mostly for technical software defects and HiL for technical hardware/software defects. Furthermore, it is important to note that apart from the evidence of reliability, performance efficiency and usability, all test types focus on the functional suitability of the test object.

In the test strategy, the test manager assigns the test scope to several different test environments. The testers should synchronize regularly to ensure efficient and accurate testing across different XiL environments. If applicable, it is recommended to align on test objectives, avoid duplication, and verify that each test is executed in the most suitable environment for its development phase. This collaboration helps prevent resource conflicts, ensures consistency in coverage, and reduces the risk of testing at inappropriate test levels. By fostering communication and leveraging shared tools or frameworks, teams can optimize test strategies and maintain a seamless progression through the testing lifecycle. By combining the criteria out of tables 3 and 4 the test manager can choose the optimal test environment.

#### 3.2.4.3 Classification of the XiL test environments in the V-model

Technical system design is on the left-hand side of the V-model. The tester can test this design with a MiL test environment. If the model and the test environment are further developed, the tester can also execute component tests and integration tests with this test environment.

The tester can use a SiL test environment if single components of the test object are programmed and compiled. Typical tests for a SiL test environment are component tests and integration tests. These can be found on the right-hand side of the V-model.

In system tests, certain functionalities of the test object have been entirely developed. The tester can execute the system test with a HiL test environment.

With a correct assignment of the test environment to the test levels, the entire test process can be optimized according to the following three aspects:

Minimizing the product risks

- Finding test level specific failure types (e.g., performance efficiency at the system level within a HiL environment)

Minimizing the test cost

- For every test level, the adequate test types are required
- Transfer of tests to earlier, less costly and virtual test levels

Conformity to standards

- In the method tables of ISO 26262, test environments are recommended depending on ASIL

# 4 Static Testing and Dynamic Testing – 230 minutes

**Keywords**

back-to-back testing, fault injection, requirements-based testing

**Learning Objectives for Chapter 4**

### 4.1 Static Testing

AuT-4.1.1 (K2) Explain the purpose and requirements of the MISRA-C guidelines with the help of examples

AuT-4.1.2 (K3) Apply requirements review using quality characteristics of the ISO/IEC/IEEE 29148 standard that are relevant to testers

### 4.2 Dynamic Testing

AuT-4.2.1 (K3) Design test cases to achieve modified condition/decision testing coverage

AuT-4.2.2 (K2) Explain the use of back-to-back testing by giving examples

AuT-4.2.3 (K2) Explain fault injection testing by giving examples

AuT-4.2.4 (K1) Recall the principles of requirements-based testing

AuT-4.2.5 (K3) Apply context dependent criteria for the choice of suitable and necessary test techniques and test approaches

# 4.1 Static Testing

## Introduction

Static testing is examining work products of software development without executing them. This includes evaluation by people performing reviews and tool-supported static analysis.

### 4.1.1 The MISRA-C Guidelines

It is the state of the art for source code to comply with coding guidelines . ISO 26262 also recommends adherence to coding guidelines for safety-relevant software[29]. Coding guidelines help avoid anomalies in the code, which could lead to defects. At the same time, they support maintainability and portability.

MISRA-C provides coding guidelines for the C programming language and is commonly used in automotive software projects. It defines two different types of guidelines: rules and directives.

- Rules are verifiable by static analysis tools. Example rule: “The source code shall not include nested comments.”
- Directives are not entirely verified by static analysis tools. They refer to details of the development process or documents outside of the software. Example directive: “The developer shall sufficiently document the implemented behavior.”

Each guideline is categorized at one of the following three levels of obligation:

- “advisory” guidelines must be followed by the developer if the effort is reasonable.
- “required” guidelines must be followed by the developer; exceptions are possible but must be officially approved (e.g., by quality management)
- “mandatory” guidelines must be followed by the developer; there are no exceptions.

Organizations may individually raise the level of obligation for a guideline but may never lower it.

### 4.1.2 Quality Characteristics for Requirements Reviews

Specifications are the basis for development and testing. Therefore, defects in specifications lead to cost and time-intensive follow up activities. This applies especially if defects are only detected in late development phases such as acceptance testing or in operation. Reviews are an effective measure to find defects in specifications early and consequently to fix them early and at low cost.

During test analysis, the tester must check the specifications for the test object, especially regarding their suitability as a test basis. Quality characteristics help the tester during the review of the specifications to focus and find as many defects as possible. ISO 29148:2018 includes quality characteristics for single requirements and for sets of requirements.

[29] See also [ISO 26262:2018] Part 6-5.4.3

Quality characteristics from ISO 29148:2018 relevant for testers include:

- Verifiable: Each requirement can be proven to be realized correctly.
- Unambiguous: Each requirement contains clear test conditions.
- Consistent: Each requirement is consistent in itself and with all other requirements.
- Complete: Each requirement does not leave any room for interpretation and is not built upon implicit knowledge or knowledge from experience.
- Singular[30]: No requirement can be divided into meaningful partial requirements.

In preparation for a requirements review, the tester can derive a review checklist with checklist items from these quality characteristics. These checklist items must be more specific than simply naming the quality criteria, which are too abstract. For example, a checklist item for consistency could be “Are terms used consistently over all requirements?”. In the review of the specification, the tester must answer the checklist items to the best of their knowledge.

According to ISO 29148:2018, requirements should also be feasible, appropriate, correct, conforming, comprehensible, and necessary. However, it is usually difficult for the tester to evaluate these quality characteristics.

[30] singular is also known as *atomic*

## 4.2 Dynamic Testing

### 4.2.1 Modified Condition/Decision Testing

The test techniques described here are part of the white-box test techniques. For further details see also syllabus for Technical Test Analyst [ISTQB_ALTTA_SYL]). The tester derives the test cases directly from the structure of the test object (e.g., source code).

In decision testing, test cases are designed to execute all possible decision outcomes, i.e., the ‘true’ and ‘false’ results of each decision. A decision consists of one or more conditions, each of which can evaluate to ‘true’ or ‘false’. The decision outcome is determined by the logical combination of these condition values.

If a decision consists of one condition, decision testing and condition testing achieve the same coverage. However, for decisions with multiple conditions, more detailed test techniques are available, which are described as follows:

- Condition testing (test technique A in table 5): The tester designs test cases with the objective of covering the true/false outcomes of each individual condition. With a poor choice of test data (see table 5), 100% condition coverage can still be achieved, but not full coverage of all the decision outcomes. For both test cases TC1 and TC2, the individual conditions B1 and B2 are exercised as both true and false, and the decision outcomes evaluate to 'false'.
- Multiple condition testing (test technique B in table 5): The tester designs test cases with the objective of covering all combinations of values related to the individual conditions. If every combination of values is tested, each decision outcome is tested as well.
- Modified condition/decision testing (MC/DC) (test technique C in table 5): This is like multiple condition testing (B). However, the test technique only considers combinations in which individual conditions (B1, B2) independently influence the decision outcome. In the case of TC4, changing either B1 or B2 from ‘false’ to ‘true’ individually does not change the decision outcome. (i.e., it remains ‘false’). 100% MC/DC coverage can be achieved by using test cases TC 1, TC 2 and TC 3; it is not necessary to consider TC 4.

Table 5 shows an example of the necessary test cases for 100% coverage depending on the test technique chosen:

| | **Individual conditions** | | **Decision outcome for the expression: E = B1 AND B2** | **Test technique** | | |
|---|---|---|---|---|---|---|
| **Test case** | B1 | B2 | | A | B | C |
| **TC 1** | B1=TRUE | B2=FALSE | E =FALSE | X | X | X |
| **TC 2** | B1=FALSE | B2=TRUE | E=FALSE | X | X | X |
| **TC 3** | B1=TRUE | B2=TRUE | E=TRUE | | X | X |
| **TC 4** | B1=FALSE | B2=FALSE | E=FALSE | | X | |

Table 5: Comparison of the test techniques condition testing (A), multiple condition testing (B) and modified condition/decision testing (MC/DC test) (C)

The example shows the limits of these test techniques: In the case of condition testing (A), despite a condition coverage of 100%, the tester takes the risk of only covering *one* decision outcome. A better choice of test cases would correct this by using test cases TC 3 and TC 4.

With the use of multiple condition testing (B) the tester can cover all possible inputs and outputs. However, the number of tests to be executed is the highest for these test techniques.

By using modified condition/decision testing (C), the tester can achieve complete coverage of all single conditions and all decisions with a fewer number of tests compared to multiple condition testing.

### 4.2.2 Back-to-Back Testing

Back-to-back testing is a test approach in which a test object is tested using a pseudo-oracle to generate expected results. To do this, the tester executes the same test case on all variants and compares the test results. If the test results are identical, the test has passed. If the test results differ, the cause of the detected difference is analyzed.

The test objects must be based on the same requirements in terms of content. Only then can they demonstrate comparable behavior. While these requirements are typically used as a basis for deriving test inputs, they do not serve as a test oracle to define the expected results. Instead, in back-to-back testing, the behavior of one test object is compared with that of another using one as a pseudo-oracle for the other. In this way, the test is expected to reveal unintended differences – even very slight ones – between the test objects or their environments.

In the simplest case, the test objects of back-to-back testing are different versions of the same software. In this case, an earlier version of the test object serves as pseudo-oracle for the back-to-back testing, like a regression test. Another alternative is the comparison of an executable model with the manual or automated generated code. In this case, it is a form of model-based testing, in which the executable model also serves as pseudo-oracle. This test approach is therefore very suitable for automated test design. Here, the tester derives not only the expected result from the model, but also automated test cases.

### 4.2.3 Fault Injection Testing

Fault injection testing is a test approach designed to assess the response of a component or system to adverse conditions (e.g. a defect - also referred to as a *fault* in this context).

Programming techniques such as error handling serve the purpose of making the system react to internal and external defects in a robust and safe way. To perform fault injection testing, the tester can selectively insert defects into the system at the following points:

- Defects in external components: e.g., detecting implausible values from sensors
- Defects in interfaces: e.g., avoiding harm from short circuits or lost messages
- Defects in the application: detecting and handling internal defects

Faults may be injected in different ways, depending on the system and test environment:

- In the classic fault injection, the tester inserts a defect by manipulating the physical test object.
- External defects or interface-related defects are typically simulated at run-time in a HiL test environment using an FIU
- Software-based defects, such as interface or internal defects, are often simulated in a SiL test environments or directly in the development environment, using tools like debuggers or the Universal Measurement and Calibration Protocol (XCP).

### 4.2.4 Requirements-Based Testing

In requirements-based testing, the tester typically analyses textual requirements to derive test conditions from individual atomic requirements, designs test cases that exercise the requirements and executes them. This decomposition involves identifying distinct, testable aspects within a requirement. Each atomic requirement should describe a single behavior or condition that can be verified independently.

By breaking down a complex or high-level requirement into such granular elements, the tester ensures comprehensive coverage and avoids missing hidden or implicit expectations. Requirements coverage is measured as the ratio of atomic requirements covered by at least one test case to the total number of atomic requirements.

Test cases for requirements-based testing are typically positive test cases, which confirm that the stated requirements are met. Negative test cases are often derived using complementary test techniques such as equivalence partitioning, error guessing, or exploratory testing, although requirements may also explicitly define such negative conditions.

### 4.2.5 Context-Dependent Selection

Several test techniques and test approaches are relevant in the context of automotive systems. Some of these are introduced in the CTFL syllabus [ISTQB_FL_SYL], while others are discussed in section (4.2). These include:

- Requirements-based testing
- Equivalence partitioning
- Boundary value analysis
- Statement testing
- Branch testing
- Modified condition/decision testing
- Error guessing
- Fault injection
- Back-to-back testing

The selection of suitable test techniques and test approaches depends on several factors:

State of the Art

Does the test technique or test approach represent the current state of the art for this purpose? Here, standards such as ISO/IEC/IEEE 29119 and ISO 26262 provide valuable guidance. Particularly ISO/IEC/IEEE 29119-4 offers a structured overview of standardized test techniques, including corresponding coverage measures, categorized into black-box test techniques, white-box test techniques, and experience-based test techniques. ISO 26262 further suggests applicable test techniques and test approaches depending on the ASIL level, as discussed in section 2.2. Deviations from the recommendations must be carefully considered and justified.

Test basis

Does the test basis provide suitable test conditions for the test technique? For example, the tester can only form equivalence partitions if the test basis includes parameters or variables. Values must be able to be grouped into equivalence partitions. Similar conditions apply to boundary values. They can only be tested if the value range is defined in a linear way.

Risk-based testing

Risk-based testing means the identification of product risks and the consideration of the risk level for the selection of the test techniques and test approaches. For example, the test of a boundary value only makes sense if there is a risk of boundary violations occurring and if the impact of such violations constitutes a risk.

Test level

Can the test technique / test approach be used on the test level? White-box testing is particularly suitable if the source code or the internal structure serves as the test basis. In the ideal case, the structural degree of coverage is measurable. For black-box testing, the test object needs to be available and observable. For example, testing of an equivalence partition of a sensor in system testing may be more efficient than in component testing. If a test technique or test approach is not usable on one test level, the tester should choose a different test level in accordance with the test strategy.

Examples of the selection of test techniques and test approaches

The following table provides a list of test techniques and test approaches, supplemented by an example of a user's assessment of several of the above factors and the choice of test technique / test approach on that basis.

| | Test technique / Test approach | Recommended for use with ASIL A? | Test basis suitable? | Risk, if defect not detected? | Test level system test reasonable? | Selection |
|---|---|---|---|---|---|---|
| **1** | Requirements-based testing | ++ | YES | ++ | YES | X |
| **2** | Equivalence partitioning | + | YES | ++ | YES | X |
| **3** | Boundary value analysis | + | NO | - | YES | |
| **4** | Statement testing | ++ | YES | ++ | NO | |
| **5** | Decision testing | + | YES | ++ | NO | |
| **6** | MC/DC | + | YES | + | NO | |
| **7** | Error guessing | + | NO | ++ | YES | |
| **8** | Fault injection testing | + | YES | + | NO | |
| **9** | Back-to-back testing | + | NO | ++ | YES | |

Table 6: Example of choice of test technique / test approach

# 5 List of Abbreviations

| Abbreviation | Definition / Meaning |
|---|---|
| ACQ | Acquisition (ASPICE) |
| ASIL | Automotive safety integrity level |
| ASAM | Association for Standardization of Automation and Measuring Systems |
| ASPICE | Automotive SPICE |
| AUTOSAR | Automotive Open System Architecture |
| AUTOSIG | Automotive Specific Interest Group |
| BP | Base Practice (ASPICE) |
| BSW | Basic Software (AUTOSAR) |
| CTFL | Certified Tester Foundation Level |
| E/E | Electric / Electronic |
| ECU | Electronic Control Unit |
| E/E | Electric/Electronic |
| EES | Electrical Error Simulation |
| EOP | End of Production |
| FIU | Fault Insertion Unit |
| GP | Generic Practice (ASPICE) |
| HiL | Hardware-in-the-loop |
| IEC | International Electrotechnical Commission |
| ISO | International Organization for Standardization |
| MAN | Management (ASPICE) |
| MC/DC | Modified Condition/Decision Coverage |
| MiL | Model-in-the-loop |

| Abbreviation | Definition / Meaning |
|---|---|
| MISRA | Motor Industry Software Reliability Association |
| OEM | Original Equipment Manufacturer |
| PA | Process Attribute (ASPICE) |
| PIM | Process Improvement (ASPICE) |
| QM | Quality Management |
| REU | Reuse (ASPICE) |
| RTE | Run Time Environment (AUTOSAR) |
| SiL | Software-in-the-loop |
| SOP | Start of Production |
| SPICE | Software Process Improvement and Capability Determination |
| SPL | Supply (ASPICE) |
| SUP | Support (ASPICE) |
| SW-C | Software Component (AUTOSAR) |
| SWE | Software Engineering (ASPICE) |
| SYS | Systems Engineering (ASPICE) |
| VAL | Validation (ASPICE) |
| VDA | German Association of the Automotive Industry |
| XCP | Universal Measurement and Calibration Protocol |
| XiL | X-in-the-loop: A general term for all in-the-loop test environments like MiL, SiL, and HiL |

# 6 Domain Specific Terms

| Term | Definition |
|---|---|
| Automotive Open System Architecture | A development partnership establishing an open industry standard for software architecture in the automotive industry. Note: The term is also used for the standardized software architecture and the associated system/software development approach. |
| automotive safety integrity level | A safety integrity level for automotive functional safety, defined in ISO 26262. |
| Automotive SPICE | A process reference model and an associated process assessment model in the automotive industry. |
| basic software (AUTOSAR) | In AUTOSAR, a software layer consisting of standardized, hardware-oriented components. |
| breakout box | A measuring unit to analyze, interrupt or manipulate physical signals in wires. |
| bus system | A network of electronic control units that exchange information via the same connection. |
| capability dimension | A dimension of a process assessment model defining the process attributes to be assessed. |
| capability indicator | An indicator for process capability used in process capability assessment. |
| capability level | A level assigned to a process based on the assessment of corresponding process attributes in a process capability assessment. |
| closed-loop system | A system in which a controlling action or an input is dependent on the output or on changes in the output. |
| directive (MISRA-C) | A programming guideline in MISRA-C that cannot be fully verified by static analysis. |
| ECU configuration description | The data required to integrate software components with an electronic control unit. |
| ECU extract | System configuration data for an electronic control unit. |
| electronic control unit | In automotive, an embedded system that controls electrical (sub)systems in a vehicle. |
| MISRA-C | A coding guideline for the use of the C programming language for critical systems provided by MISRA. |
| open-loop system | A system in which a controlling action or an input is independent of the output or of changes in the output. |
| original equipment manufacturer | In the automotive industry, a car manufacturer. |

| Term | Definition |
|---|---|
| process attribute | A set of measurable characteristics of a process for a process capability assessment. |
| process dimension | A dimension of a process assessment model defining the processes to be assessed. |
| product development process | A process that includes all activities from first product idea until production. |
| real time | The processing of data so that results are available within a predetermined time period. |
| real time capable computer | A computer that is capable of processing signals in real time. |
| release | Documentation of releasing a release item. |
| release item | An identifiable element having implemented functions, properties and intended use. |
| release process | A process that leads to a release. |
| release recommendation | A recommendation whether to release a release item based on the test results. |
| rule (MISRA-C) | A programming guideline in MISRA-C that is fully verifiable by static analysis. |
| runtime environment (AUTOSAR) | In AUTOSAR, the abstraction layer which controls and implements the data exchange between AUTOSAR software components as well as between application software and basic software, both within an electronic control unit and between electronic control units. |
| safety lifecycle | The lifecycle of a safety relevant system from inception to disposal. |
| simulation time | The time frame of a computer simulation. |
| software component (AUTOSAR) | In AUTOSAR, a component in the hardware-independent application software layer. |
| software component verification and integration verification (ASPICE) | In Automotive SPICE, a verification of the integrated software based on the software architectural design. |
| software unit verification (ASPICE) | In Automotive SPICE, a verification of the software units for consistency with their detailed design |
| software verification (ASPICE) | In Automotive SPICE, a verification of the integrated software for consistency with the software requirements |
| system configuration description | The data used in the integration of all electronic control units in a vehicle. |
| system integration and integration verification (ASPICE) | In Automotive SPICE, a verification of the integrated system elements for consistency with the system architecture. |

| Term | Definition |
|---|---|
| system lifecycle | The phases of development of a system up to its retirement. |
| system verification (ASPICE) | In Automotive SPICE, a verification of the system for consistency with the system requirements |

# 7 References

## 7.1 Standards


| | |
|---|---|
| **Automotive SPICE 4.0** | (2023), Automotive SPICE Process Assessment / Reference Model |
| **ISO 14229** | (2020), Road vehicles – Unified diagnostic services (UDS) |
| **ISO 26262** | (2018), Road vehicles – Functional safety |
| **ISO 26262-1** | (2018), Road vehicles – Functional safety<br>Part 1: Vocabulary |
| **ISO 26262-2** | (2018), Road vehicles – Functional safety<br>Part 2: Management of functional safety |
| **ISO 26262-3** | (2018), Road vehicles – Functional safety<br>Part 3: Concept phase |
| **ISO 26263-4** | (2018), Road vehicles – Functional safety<br>Part 4: Product development at the system level |
| **ISO 26263-5** | (2018), Road vehicles – Functional safety<br>Part 5: Product development at the hardware level |
| **ISO 26263-6** | (2018), Road vehicles – Functional safety<br>Part 6: Product development at the software level |
| **ISO 26263-7** | (2018), Road vehicles – Functional safety<br>Part 7: Production, operation, service and decommissioning |
| **ISO 26263-8** | (2018), Road vehicles – Functional safety<br>Part 8: Supporting processes |
| **ISO 26263-9** | (2018), Road vehicles – Functional safety<br>Part 9: Automotive safety integrity level (ASIL)-oriented and safety-oriented analyses |
| **ISO 26263-10** | (2018), Road vehicles – Functional safety<br>Part 10: Guidelines on ISO 26262 |
| **ISO/IEC 25010** | (2023), Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE)<br>Product quality model |
| **ISO/IEC 33020** | (2019), Information technology – Process assessment<br>Process measurement framework for assessment of process capability |
| **ISO/IEC/IEEE 12207** | (2017), Systems and software engineering – Software life cycle processes |
| **ISO/IEC/IEEE 15288** | (2023), Systems and software engineering – System life cycle processes |
| **ISO/IEC/IEEE 24748-1** | (2018), Systems and software engineering – Life cycle management<br>Part 1: Guide for life cycle management |

**ISO/IEC/IEEE 29119-1** (2022), Software and systems engineering – Software testing Part 1: General concepts

**ISO/IEC/IEEE 29119-2** (2021), Software and systems engineering – Software testing Part 2: Test processes

**ISO/IEC/IEEE 29119-3** (2021), Software and systems engineering – Software testing Part 3: Test documentation

**ISO/IEC/IEEE 29119-4** (2021), Software and systems engineering – Software testing Part 4: Test techniques

**ISO/IEC/IEEE 29148** (2018), Systems and software engineering – Life cycle processes – Requirements engineering

**MISRA C** (2025), Guidelines for the use of the C language in critical systems

## 7.2 ISTQB® Documents

**[ISTQB_ALTTA_SYL]** ISTQB® Advanced Level Technical Test Analyst Syllabus, Version 2012

**[ISTQB_FL_SYL]** ISTQB® Foundation Level Syllabus v4.0, 2023

## 7.3 Glossary References

References for terminology used in this Syllabus:

**IREB® Glossary** https://cpre.ireb.org/en/downloads-and-resources/glossary

**ISTQB® Glossary** https://glossary.istqb.org/

# 8 Trademarks

CTFL® is a registered trademark of the German Testing Board (GTB) e. V. in the EU only

GTB® is a registered trademark of the German Testing Board (GTB) e. V. in the EU only

ISTQB® is a registered trademark of the International Software Testing Qualifications Board

Automotive SPICE® is a registered trademark of the German Association of the Automotive Industry (VDA)

# 9 Appendix A – Learning Objectives/Cognitive Level of Knowledge

The specific learning objectives applying to this syllabus are shown at the beginning of each chapter. Each topic in the syllabus will be examined according to the learning objective for it.

The learning objectives begin with an action verb corresponding to its cognitive level of knowledge as listed below.

## *Level 1: Remember (K1)*

The candidate will remember, recognize and recall a term or concept.

**Action verbs:** Recall, recognize.

| **Examples** |
|---|
| Recall the concepts of the test pyramid. |
| Recognize the typical objectives of testing. |

## *Level 2: Understand (K2)*

The candidate can select the reasons or explanations for statements related to the topic, and can summarize, compare, classify and give examples for the testing concept.

**Action verbs**: Classify, compare, differentiate, distinguish, explain, give examples, interpret, summarize

| **Examples** | **Notes** |
|---|---|
| Classify test tools according to their purpose and the test activities they support. | |
| Compare the different test levels. | Can be used to look for similarities, differences or both. |
| Differentiate testing from debugging. | Looks for differences between concepts. |
| Distinguish between project and product risks. | Allows two (or more) concepts to be separately classified. |
| Explain the impact of context on the test process. | |
| Give examples of why testing is necessary. | |
| Infer the root cause of defects from a given profile of failures. | |
| Summarize the activities of the work product review process. | |

## *Level 3: Apply (K3)*

The candidate can carry out a procedure when confronted with a familiar task or select the correct procedure and apply it to a given context.

**Action verbs**: Apply, implement, prepare, use

| Examples | Notes |
|---|---|
| Apply boundary value analysis to derive test cases from given requirements. | Should refer to a procedure / technique / process etc. |
| Implement metric collection methods to support technical and management requirements. | |
| Prepare installability tests for mobile apps. | |
| Use traceability to monitor test progress for completeness and consistency with the test objectives, test strategy, and test plan. | Could be used in a LO that wants the candidate to be able to use a technique or procedure. Like 'apply'. |

## Reference

(For the cognitive levels of learning objectives)

Anderson, L. W. and Krathwohl, D. R. (eds) (2001) A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Allyn & Bacon

# 10 Appendix B – Business Outcomes Traceability Matrix with Learning Objectives

This chapter lists the traceability between the Business Outcomes and the Learning Objectives of Certified Tester Automotive Software Tester.

| Business Outcomes: Automotive Software Tester | | | BO1 | BO2 | BO3 | BO4 | BO5 |
|---|---|---|---|---|---|---|---|
| AuT-BO1 | Collaborate effectively in a test team. (“collaborate”) | | 37 | | | | |
| AuT-BO2 | Adapt the test techniques known from the ISTQB® Certified Tester Foundation level (CTFL®) to the specific project requirements. (“adapt”) | | | 7 | | | |
| AuT-BO3 | Consider the basic requirements of the relevant standards (i.e., Automotive SPICE® and ISO 26262) for the selection of suitable test techniques. (“select”) | | | | 27 | | |
| AuT-BO4 | Support the test team in the risk-based planning of the test activities and apply known elements of structuring and prioritization. (“support & apply”) | | | | | 9 | |
| AuT-BO5 | Apply the virtual test environments (i.e., MiL, SiL, and HiL) in test environments. (“apply”) | | | | | | 13 |

| Business Outcomes: Automotive Software Tester | | | BO1 | BO2 | BO3 | BO4 | BO5 |
|---|---|---|---|---|---|---|---|
| Unique LO | Learning Objective | K-Level | | | | | |
| 1 | Introduction | | | | | | |
| 1.1 | Requirements from divergent project objectives and increasing product complexity | | | | | | |
| AuT-1.1 | Explain and give examples of the challenges of automotive product development that arise from divergent project objectives and increasing product complexity | K2 | X | | | | |
| 1.2 | Project Aspects Influenced by Standards | | | | | | |
| AuT-1.2 | Recall project aspects that are influenced by standards (e.g., time, cost, quality and project risks and product risks) | K1 | X | | | | |
| 1.3 | The Six Generic Stages in the System Lifecycle | | | | | | |
| AuT-1.3 | Recall the six generic stages in the system lifecycle per ISO/IEC/IEEE 24748-1 | K1 | X | | | | |

| Business Outcomes: Automotive Software Tester | | | BO1 | BO2 | BO3 | BO4 | BO5 |
|---|---|---|---|---|---|---|---|
| 1.4 | The Contribution and Participation of the Tester in the Release Process | | | | | | |
| AuT-1.4 | Recall the role (i.e., contribution and collaboration) of the tester in the release process | K1 | X | | | | |
| 2 | Standards for the Testing of Electric/Electronic (E/E) Systems | | | | | | |
| 2.1 | Automotive SPICE (ASPICE) | | | | | | |
| AuT-2.1.1.1 | Recall the two dimensions of Automotive SPICE (ASPICE) | K1 | | | X | | |
| AuT-2.1.1.2 | Recall the process categories and process groups of ASPICE [informative] | K1 | | | X | | |
| AuT-2.1.1.3 | Explain the Capability levels 0 to 3 of ASPICE | K2 | | | X | | |
| AuT-2.1.2.1 | Recall the purpose of the test-specific relevant processes of ASPICE | K1 | X | | X | | |
| AuT-2.1.2.2 | Explain the meaning of the four rating levels and the capability indicators of ASPICE from the testing perspective | K2 | | | X | | |

| Business Outcomes: Automotive Software Tester | | | BO1 | BO2 | BO3 | BO4 | BO5 |
|---|---|---|---|---|---|---|---|
| AuT-2.1.2.3 | Explain the requirements of ASPICE for the test strategy including the criteria for regression verification | K2 | X | | X | X | |
| AuT-2.1.2.4 | Recall the requirements of ASPICE for the testware | K1 | X | | X | X | |
| AuT-2.1.2.5 | Use verification measures for software unit verification | K3 | X | | X | X | |
| AuT-2.1.2.6 | Explain the different traceability requirements of ASPICE from the testing perspective | K2 | X | | X | | |
| 2.2 | ISO 26262 | | | | | | |
| AuT-2.2.1.1 | Explain the objective of functional safety for E/E systems | K2 | | | X | | |
| AuT-2.2.1.2 | Recall the testers' contribution for the safety culture | K1 | X | | X | | |
| AuT-2.2.2 | Discuss the role of the tester in the framework of the safety lifecycle per ISO 26262 | K2 | X | | X | | |
| AuT-2.2.3.1 | Recall the design and structure of ISO 26262 [informative] | K1 | | | X | | |
| AuT-2.2.3.2 | Recall the parts of ISO 26262 that are relevant to the tester | K1 | X | | X | | |

| Business Outcomes: Automotive Software Tester | | | BO1 | BO2 | BO3 | BO4 | BO5 |
|---|---|---|---|---|---|---|---|
| AuT-2.2.4.1 | Recall the criticality levels of ASIL | K1 | X | | X | | |
| AuT-2.2.4.2 | Explain the influence of ASIL on applicable test techniques and test types for static testing and dynamic tests and the resulting test scope | K2 | X | X | X | X | |
| AuT-2.2.5 | Use the method tables of ISO 26262 | K3 | X | X | X | X | |
| 2.3 | AUTOSAR | | | | | | |
| AuT-2.3.1 | Recall the project objectives of AUTOSAR | K1 | | | X | | |
| AuT-2.3.2 | Recall the general design of AUTOSAR [informative] | K1 | | | X | | |
| AuT-2.3.3 | Recall the influence of AUTOSAR on the work of the tester | K1 | X | X | X | | |
| 2.4 | Comparison of ASPICE, ISO 26262, and CTFL® | | | | | | |
| AuT-2.4.1 | Recall the different objectives of ASPICE and ISO 26262 | K1 | | | X | | |
| AuT-2.4.2 | Explain the differences between ASPICE and ISO 26262 and CTFL® regarding test levels | K2 | X | X | X | | |

| Business Outcomes: Automotive Software Tester | | | BO1 | BO2 | BO3 | BO4 | BO5 |
|---|---|---|---|---|---|---|---|
| 3 | Testing in a Virtual Environment | | | | | | |
| 3.1 | Test Environment in General | | | | | | |
| AuT-3.1.1 | Recall the motivation behind a test environment in automotive development | K1 | X | | | | X |
| AuT-3.1.2 | Recall the general parts of an automotive specific test environment | K1 | X | | | | X |
| AuT-3.1.3 | Recall the differences between closed-loop systems and open-loop systems | K2 | X | | | | X |
| AuT-3.1.4 | Recall the essential functions, databases and Protocols of an ECU | K1 | X | | | | X |
| 3.2 | Testing in XiL Test Environments | | | | | | |
| AuT-3.2.1.1 | Recall the structure of a MiL test environment | K1 | X | | | | X |
| AuT-3.2.1.2 | Explain the application area and the boundary conditions of a MiL test environment | K2 | X | | | | X |
| AuT-3.2.2.1 | Recall the structure of a SiL test environment | K1 | X | | | | X |

| Business Outcomes: Automotive Software Tester | | | BO1 | BO2 | BO3 | BO4 | BO5 |
|---|---|---|---|---|---|---|---|
| AuT-3.2.2.2 | Recall the application areas and the boundary conditions of a SiL test environment | K1 | X | | | | X |
| AuT-3.2.3.1 | Recall the structure of a HiL test environment | K1 | X | | | | X |
| AuT-3.2.3.2 | Explain the application areas and the boundary conditions of a HiL test environment | K2 | X | | | | X |
| AuT-3.2.4.1 | Summarize the advantages and disadvantages of testing using criteria for the XiL test environments | K2 | X | | | X | X |
| AuT-3.2.4.2 | Apply criteria for the assignment of a given extent of the test to one or more test environments | K3 | X | | | X | X |
| AuT-3.2.4.3 | Outline the XiL test environments in the V-model | K1 | X | | | X | X |
| 4 | Static Testing and Dynamic Testing | | | | | | |
| 4.1 | Static Testing | | | | | | |
| AuT-4.1.1 | Explain the purpose and requirements of the MISRA-C guideline with the help of examples | K2 | X | X | | | |
| AuT-4.1.2 | Apply requirements review using the quality characteristics of the ISO 29148:2018 standard that are relevant to testers | K3 | X | X | | | |

| Business Outcomes: Automotive Software Tester | | | BO1 | BO2 | BO3 | BO4 | BO5 |
|---|---|---|---|---|---|---|---|
| 4.2 | Dynamic Testing | | | | | | |
| AuT-4.2.1 | Design test cases to achieve modified condition/decision testing coverage | K3 | X | | X | | |
| AuT-4.2.2 | Explain the use of back-to-back testing by giving examples | K2 | X | | X | | |
| AuT-4.2.3 | Explain fault injection testing by giving examples | K2 | X | | X | | |
| AuT-4.2.4 | Recall the principles of requirements-based testing | K1 | X | | X | | |
| AuT-4.2.5 | Apply context dependent criteria for the choice of suitable and necessary test techniques and test approaches | K3 | X | X | X | X | |

# 11 Appendix C – Release Notes

The v2.1.1 Errata version includes the following changes:

- 2.1.2.1: Added footnote for VAL process group also being test-related
- 2.1.2.2: renamed “generic […] resources” to “information items”
- 2.1.2.4: ID of information item “Verification Results” replaced by “15-52”
- 2.4.2, Table 2: Improved to better reflect the mapping of ASPICE and ISO 26262 to ISTQB terms on the component (integration) test levels, added two footnotes in the first column
- 4.1.2: enhanced definition of “consistent” to also cover a single requirement
- Appendix B: inconsistencies with learning objectives 3.1.4 and 4.2.5 fixed

**Release Notes for v2.1**

The ISTQB® Automotive Software Tester Syllabus v2.1 (2025) is a minor update of v2.0.1 (2017). The learning objectives themselves are unchanged, but the content of some learning objectives was updated to reflect updated standards referenced in the syllabus. Additionally, changes between the ISTQB CTFL 3.1 and CTFL 4.0 and updates in the ISTQB Glossary were considered. Also, consistency and understandability were improved when necessary.

The following updated standards were taken into account:

- ISO 24748-1:2024
- ISO 26262:2018
- Automotive SPICE 4.0
- AUTOSAR Classic Release R23-11
- MISRA-C:2025
- ISO 29148:2018

The naming scheme of the learning objectives changed to “AuT” plus section number.

# 12 Appendix D – Automotive Databases and Communication Protocols

| Interfaces | Database | Communication protocols |
|---|---|---|
| Memory | ASAM MCD-2 MC<br>(also ASAP2 or A2L) | ASAM MCD-1 XCP<br>(Universal Measurement and Calibration Protocol)<br><br>ASAM MCD-1 CCP<br>(CAN Calibration Protocol) |
| Bus | ASAM MCD-2 NET standard<br>(also *FIBEX - Field Bus Exchange Format*) | FlexRay<br>(ISO 17458)<br><br>CAN<br>(Controller Area Network per ISO 11898-2) |
| | DBC<br>(communication database for CAN) | CAN<br>(Controller Area Network per ISO 11898-2) |
| Diagnostics | ASAM MCD-3 D<br>(Application Programming Interface Specification)<br><br>ASAM SOVD<br>(Service-Oriented Vehicle Diagnostics)<br><br>CDD<br>(CANdelaStudio diagnostic description) | KWP2000 (ISO 14230)<br><br>ISO-OBD (ISO 15031)<br><br>UDS (ISO 14229*)*<br><br>SOME/IP (AUTOSAR Specification) |

Table 6: Common databases and communication protocols from the automotive industry

AUTOSAR has a standardized XML format which integrates the databases of a complete vehicle. It is called ARXML format (AUTOSAR Integrated Master Table of Application Interfaces, XML scheme R21-11).

# 13 Index