




FEBRUARY 8, 2021

# SOFTWARE TESTING COMPENDIUM

v2021.01

ÖMER KARACAN  
SW-T-(IT 538)  
Sabanci University



# SOFTWARE TESTING COMPENDIUM

## Table of Contents

1	Introduction.....	4
1.1	Objectives of Testing .....	5
1.2	Testing’s Contributions to Success .....	5
1.3	Errors, Defects, Failures, Root Causes and Effects .....	5
1.4	Seven Testing Principles .....	6
1.5	Supplementary Material.....	6
1.6	Exercises .....	6
2	Software Development and Testing .....	8
2.1	Software Development Lifecycle Models .....	8
2.2	General V-Model .....	8
2.2.1	Basic Characteristic.....	9
2.2.2	Supplementary material .....	10
2.3	Agile Method .....	10
2.4	General V-Model Documentation View .....	11
2.5	Supplementary Material.....	12
2.6	Exercises .....	13
3	Concepts and Definitions.....	14
3.1	Fundamental Test Process.....	14
3.2	Test Work Products .....	16
3.3	Traceability .....	17
3.4	Test Levels .....	17
3.5	Test Types.....	22
3.6	Supplementary Material.....	24
3.7	Exercises .....	25
4	Static Testing.....	27
4.1	Characteristics .....	27
4.2	Supplementary Material.....	28
4.3	Exercises .....	28
5	Test Techniques.....	29
5.1	Black-box Test Techniques .....	29
5.1.1	Equivalence Partitioning .....	30
5.1.2	Boundary Value Analysis .....	31

5.1.3	State Transition Testing .....	32
5.1.4	Use Case Testing .....	34
5.2	White-box Test Techniques .....	36
5.3	Supplementary Material.....	38
5.4	Exercises .....	39
6	Test Tools.....	40
6.1	Test Tools Basics .....	40
6.2	Supplementary Material.....	41
6.3	Exercises .....	41
7	Risks and Testing.....	42
7.1	Risk Definition.....	42
7.2	Product Risks .....	42
7.3	Project Risks.....	42
7.4	Risk-based Testing and Product Quality .....	43
7.5	Risk Management procedure .....	44
8	References .....	46
9	ANNEX – TOPIC SECTION .....	47
9.1	Overview or 1st sub-topic if given .....	47
9.2	2nd sub-topic if given .....	47
9.3	Exercises .....	47
9.4	Supplementary Material.....	47

## History of Change

26.05.2020	v2020.01	<ul style="list-style-type: none"> <li>Initial creation</li> </ul>
11.06.2020	v2020.02	<ul style="list-style-type: none"> <li>Figure ISTQB FUNDAMENTAL TEST PROCESS replaced w/ a detailed one!</li> <li>1 line added: "Test implementation work products include ..."</li> <li>1 line added: "Test execution work products ..."</li> <li>2.5 Supplementary material revisited.</li> <li>Exercise 2.6 is extended.</li> </ul>
21.06.2020	v2020.03	<ul style="list-style-type: none"> <li>Supplementary Material is inserted into Ch.2-3 and 6</li> <li>Figure 7 updated</li> <li>COMPENDIUM file name convention is changed: usage of version number instead of date!</li> <li>Supplementary Material section updated!</li> <li>"Dynamic Test Process" is defined and Figure 4 updated</li> <li>Figure 5 inserted into 3.3</li> <li></li> </ul>
30.06.2020	V2020.04	<ul style="list-style-type: none"> <li>Figures are provided for ch.5</li> </ul>
08.02.2021	V2021.01	<ul style="list-style-type: none"> <li>Precise referencing to the corresponding chapter in [ISTQB2018] to ease the creation of the presentation slides per chapter.</li> </ul>

**Important remark for readers:** In this document, the quoted text is surrounded by quotation marks and in italics, e.g., *“quoted-text”*. In-text citations are used directly after the quoted text, e.g., *“quoted-text [SWT2012]”*. If a citation is missing than it is intentional and purposely, which means that the text is without exception identical to the standard document “ISTQB Certified Tester Foundation Level Syllabus Version 2018 V3.1” or [ISTQB2018].

**Important remark for students:** Select a concrete system of your preference and use it as a context (reference) for the exercises found at the end of selected chapters.

# 1 Introduction

ISTQB 2018 Syllabus Ch. 1 Fundamentals of Testing  
 ISTQB 2018 Syllabus Ch. 1.1 What is Testing?

“Software systems are an integral part of life, from business applications (e.g., banking) to consumer products (e.g., cars). Most people have had an experience with software that did not work as expected.

Software that does not work correctly can lead to many problems, including loss of money, time, or business reputation, and even injury or death.

Software testing is a way to assess the quality of the software and to reduce the risk of software failure in operation.”

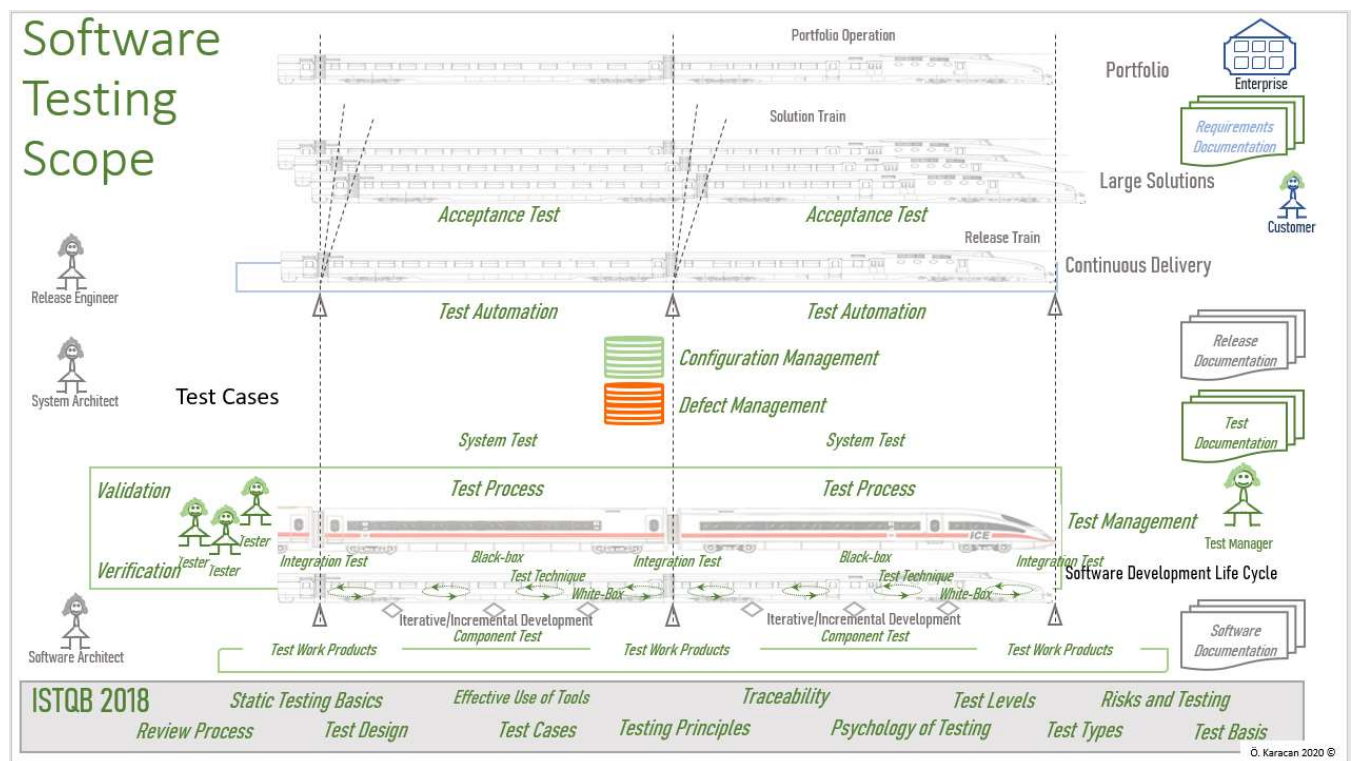


FIGURE 1 SOFTWARE TESTING SCOPE

“Some testing does involve the execution of the component or system being tested; such testing is called dynamic testing.”

Dynamic testing

“Other testing does not involve the execution of the component or system being tested; such testing is called static testing. So, testing also includes reviewing work products such as requirements, user stories, and source code.”

Static testing  
 Review

Software testing focuses on “verification of requirements, user stories, or other specifications”, i.e. “whether the system meets specified requirements” and on “validation, which is checking whether the system will meet user and other stakeholder needs in its operational environment(s).”

Verification<sup>1</sup> and validation<sup>2</sup> of requirements

## 1.1 Objectives of Testing

ISTQB 2018 Syllabus Ch. 1.1.1 Typical Objectives of Testing

“For any given project, the objectives of testing may include:

- To prevent defects by evaluate work products such as requirements, user stories, design, and code,
- To verify whether all specified requirements have been fulfilled,
- To check whether the test object is complete and validate if it works as the users and other stakeholders expect,
- To build confidence in the level of quality of the test object,
- To find defects and failures thus reduce the level of risk of inadequate software quality,
- To provide sufficient information to stakeholders to allow them to make informed decisions, especially regarding the level of quality of the test object,
- To comply with contractual, legal, or regulatory requirements or standards, and/or to verify the test object’s compliance with such requirements or standards.

## 1.2 Testing’s Contributions to Success

ISTQB 2018 Syllabus Ch. 1.2.1 Testing’s Contributions to Success

“Rigorous testing of components and systems, and their associated documentation, can help reduce the risk of failures occurring during operation.” In this context, the testers review and analysis the test objects intensively in all phases of development. For example:

“Having testers involved in requirements reviews or user story refinement could detect defects in these work products.”

Requirements analysis

“Having testers work closely with system designers while the system is being designed can increase each party’s understanding of the design and how to test it” and thus “can reduce the risk of fundamental design defects.”

System software design

“Having testers work closely with developers while the code is under development can increase each party’s understanding of the code and how to test it” and thus “can reduce the risk of defects within the code and the tests.”

Software implementation

“Having testers verify and validate the software prior to release can detect failures that might otherwise have been missed and support the process of removing the defects that caused the failures.”

Component testing  
System testing

## 1.3 Errors, Defects, Failures, Root Causes and Effects

ISTQB 2018 Syllabus Ch. 1.2.3 Errors, Defects, and Failures

ISTQB 2018 Syllabus Ch. 1.2.4 Defects, Root Causes and Effects

<sup>1</sup> Verification: “Confirmation by examination and through provision of objective evidence that specified requirements have been fulfilled.” [ISTQBterm]

<sup>2</sup> Validation: “Confirmation by examination and through provision of objective evidence that the requirements for a specific intended use or application have been fulfilled.” [ISTQBterm]

*“A person can make an error (mistake), which can lead to the introduction of a defect (fault or bug) in the software code or in some other related work product.*

*If a defect in the code is executed, this may cause a failure”.* The failure is the perception of the defects outside of the system. That is, an error injected by a programmer into the software is a defect. When a defect is executed, it may cause the software malfunction. The customer’s perception is a failure of the system, and his/her complaint is an effect.

*“For example, suppose incorrect interest payments, due to a single line of incorrect code, result in customer complaints.*

*The defective code was written for a user story which was ambiguous, due to the product owner’s misunderstanding of how to calculate interest.*

*In this example, the customer complaints are effects. The incorrect interest payments are failures. The improper calculation in the code is a defect, and it resulted from the original defect, the ambiguity in the user story.*

*The root cause of the original defect was a lack of knowledge on the part of the product owner, which resulted in the product owner making an error while writing the user story.”*

Example

Error, defect

Effect

Failure

Root cause

## 1.4 Seven Testing Principles

*ISTQB 2018 Syllabus Ch. 1.3 Seven Testing Principles*

There are unwritten principles of software testing, which may be valid for testing all kinds of testing, which are:

1. *Testing shows the presence of defects, not their absence,*
2. *Exhaustive testing is impossible,*
3. *Early testing saves time and money,*
4. *Defects cluster together,*
5. *Beware of the pesticide paradox,*
6. *Testing is context dependent,*
7. *Absence-of-errors is a fallacy (i.e., a mistaken belief)*

## 1.5 Supplementary Material

- Course Video Clip
  - Name: “What is Software Testing & Why Testing is Important?”
  - Location: <https://youtu.be/TDynSmrzpXw>
- Course Video Clip
  - Name: “Errors Defects and Failures”
  - Location: <https://www.youtube.com/watch?v=CxBxQEGcYKk>
- Course Video Clip
  - Name: “Seven Testing Principles: Software Testing”
  - Location: <https://www.youtube.com/watch?v=rFaWOw8bIMM>

## 1.6 Exercises

1. *Distinguish between error, defect, and failure. Give two examples from your selected system.*
2. *Distinguish between the root cause of a defect and its effects. Give two examples from your selected system.*

3. *Explain two of your choice of the seven testing principles. Give an example from your selected system.*
4. *Explain the difference between "Verification" and "Validation" of requirements.*



## 2 Software Development and Testing

### 2.1 Software Development Lifecycle Models

*ISTQB 2018 Syllabus Ch. 2.1 Software Development Lifecycle Models*

*“A software development lifecycle model describes the types of activity performed at each stage in a software development project, and how the activities relate to one another logically and chronologically.”*

*“In any software development lifecycle model, there are several characteristics of good testing:*

- *For every development activity, there is a corresponding test activity,*
- *Each test level has test objectives specific to that level,*
- *Test analysis and design for a given test level begin during the corresponding development activity,*
- *Testers participate in discussions to define and refine requirements and design, and are involved in reviewing work products (e.g., requirements, design, user stories, etc.) as soon as drafts are available.*

*No matter which software development lifecycle model is chosen, test activities should start in the early stages of the lifecycle.”*

*“Software development lifecycle models must be selected and adapted to the context of project and product characteristics. An appropriate software development lifecycle model should be selected and adapted based on the project goal, the type of product being developed, business priorities (e.g., time-to-market), and identified product and project risks.”*

Today, software development projects use mainly three types of lifecycle model as follows:

- Sequential development lifecycle model,
- Iterative and incremental development lifecycle model, and
- Hybrid models as a combination of both.

*As a sequential model, the “V-model integrates the test process throughout the development process, implementing the principle of early testing. Further, the V-model includes test levels associated with each corresponding development phase, which further supports early testing .... In this model, the execution of tests associated with each test level proceeds sequentially, but in some cases overlapping occurs.”*

V-model

*As an Iterative and Incremental development model, the “Agile development involves small iterations of software design, build, and test that happen on a continuous basis, supported by on-going planning. So, test activities are also happening on an iterative, continuous basis within this software development approach.”*

Agile model

The hybrid models are usually company specific models that are designed to satisfy product specific expectations. Usually, the agility is given on team level and above teams the project structure and management are shaped by the company’s traditional development methodology. The hybrid model will not be debated further in this document.

Hybrid model

### 2.2 General V-Model

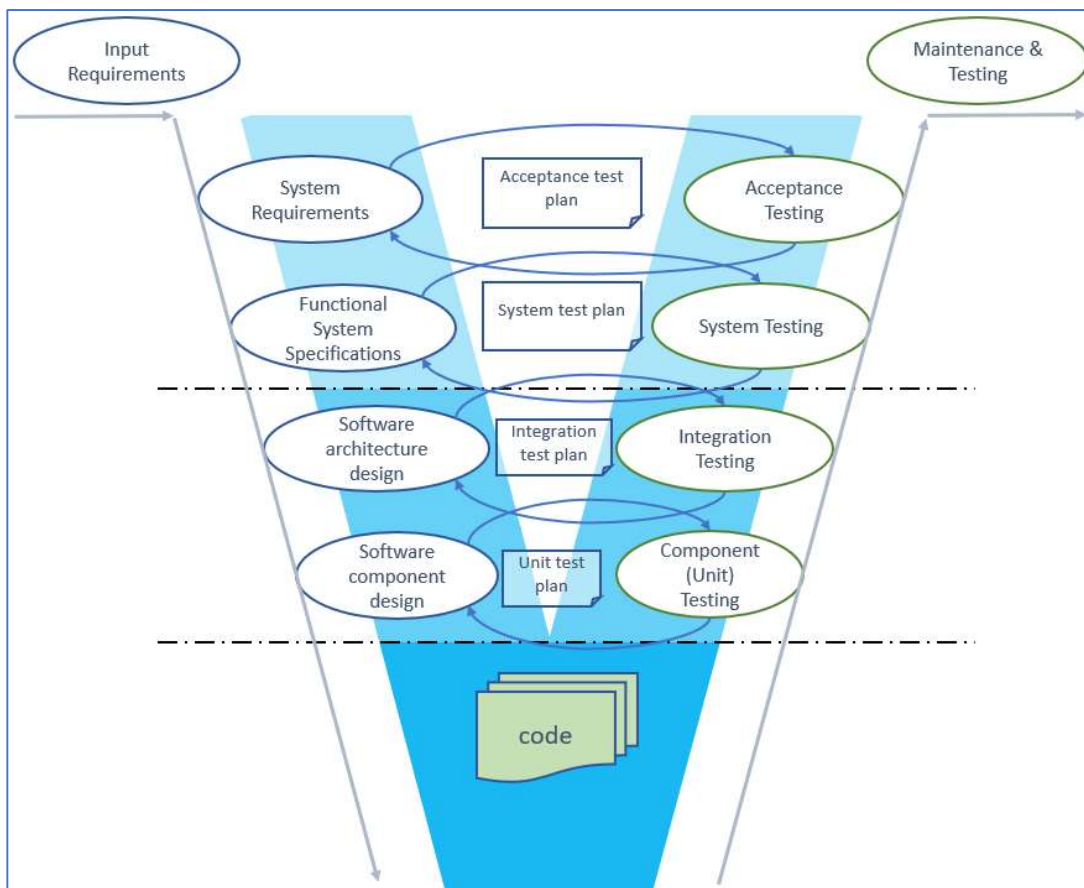
*Text in italics: [STFebook]*

### 2.2.1 Basic Characteristics

*“An enhancement of the waterfall model is the general V-model, where the constructive activities are decomposed [(separated)] from the testing activities. The model has the form of a V. The constructive activities, from requirements definition to implementation, are found on the downward branch of the V [left side]. The test execution activities on the ascending branch are organized by test levels and matched to the appropriate abstraction level on the opposite side’s constructive activity. “*

The basic characteristic of the general V-model is the separation of development and testing tasks as corresponding activities, that is, on each level, specification, planning and testing activities complete a level-specific (horizontal) development life-cycle.

Testing is planned and executed at all levels i.e. input from requirements to the maintenance of the delivered software. The two gradients of “V” illustrate this concept.



**FIGURE 2 GENERAL V-MODEL – PROCESS VIEW**

The left gradient represents the software construction, that is, development activities without validation and verification, but “top-down and upstream tracing”<sup>3</sup>.

<sup>3</sup> Top-down and upstream tracing of the requirements is a method of verification (based on expert review) whether the requirements are considered completely at all levels down to the code, and vice versa.

During development, the input requirements are analyzed, systems requirements are identified, functional system specifications and software architecture are created, architecture components are designed and finally coded.

The right gradient of the V-model defines corresponding test levels, that is, *“for each specification and construction level, the right branch of the V-model defines a corresponding test level:”*

- *Component test verifies whether each software component correctly fulfills its specification.*
- *Integration test checks if groups of components interact in the way that is specified by the technical system design.*
- *System test” validates “whether the system as a whole meets the specified requirements.*
- *Acceptance test checks if the system meets the customer requirements, as specified in the contract and/or if the system meets user needs and expectations.”*

### 2.2.2 Supplementary material

- Course Video Clip
  - Name: “What Are the Steps of the Software Development Lifecycle?”
  - Location: <https://www.youtube.com/watch?v=gNmrGZSGK1k>
- Course Video Clip
  - Name: “SDLC Vs STLC”
  - Location: <https://youtu.be/An7HC1LoIDM> or embedded in <https://www.guru99.com/v-model-software-testing.html>

## 2.3 Agile Method

Text in italics: <https://www.guru99.com/agile-scrum-extreme-testing.html>

*“Agile method proposes incremental and iterative approach to software design.”*

Based on Extreme Programming, *“Agile methodology is a practice that promotes continuous iteration of development and testing throughout the software development lifecycle of the project. Both development and testing activities are concurrent.”*

Agile The characteristics of the Agile Method are:

- *“Agile method proposes incremental and iterative approach to software design*
- *The agile process is broken into individual models that designers work on*
- *The customer has early and frequent opportunities to look at the product and make decision and changes to the project*
- *Agile model is considered unstructured compared to the waterfall model*
- *Small projects can be implemented very quickly. For large projects, it is difficult to estimate the development time.*
- *Error can be fixed in the middle of the project.*
- *Development process is iterative, and the project is executed in short (2-4) weeks iterations. Planning is very less*
- *Documentation attends less priority than software development*
- *Every iteration has its own testing phase. It allows implementing regression testing every time new functions or logic are released.*
- *In agile testing when an iteration end, shippable features of the product is delivered to the customer. New features are usable right after shipment. It is useful when you have good contact with customers.*
- *Testers and developers work together*
- *At the end of every sprint, user acceptance is performed*

- *It requires close communication with developers and together analyze requirements and planning.”*

At the beginning of Agile hype, it was a guidance for development teams individually. As the agility becomes popular a need is emerged for orchestration of large numbers of agile teams in an enterprise. The Scaled Agile Framework (SAFe) defines a set of organizational and workflow patterns for enterprises to scale lean and agile practices. The SAFe supports alignment, coordination, and budget-aware product release across large numbers of agile teams.

### 2.4 General V-Model Documentation View

Depending on the project type, the document landscape changes, for example, safety critical projects are document driven projects, because the software as a product underlies stringent international safety rules. Contrarily, a documentation management cloud application like dropbox may need few documents to drive the development.

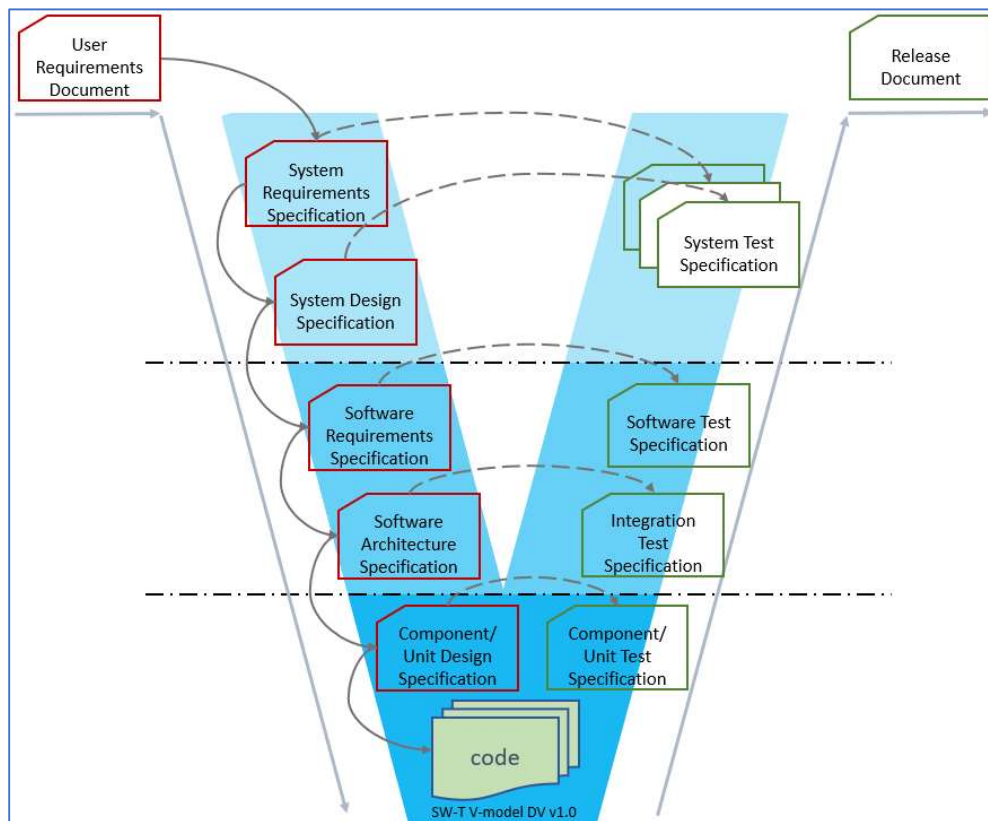


FIGURE 3 GENERAL V-MODEL – DOCUMENTATION VIEW

The **User Requirements Specification** (CRS) describes end user expectations, is drafted from the client’s point of view, and focuses on business aspects rather than the technological ones. It is more or less a wish list.

The **System Requirements Specification** (SRS-SYS) document describes all data, functional and non-functional requirements of the system under development, which is written from the user’s perspective. This document has a legal character of business agreement with the customer.

The **System Design Specification** (SDS) describes the complete design for the system under development, which is written toward the architects and developers who will use it to create the necessary components and test use cases.

- Validation/Verification method is embedded in the **System Test Specification**, which describes the system (level) test cases to be performed to validate that the system works according to SRS-SYS. System (level) test cases describe what to test (functional and non-functional requirements, system interfaces and data sets, and quality attributes) and how to test (test scenarios, input test data, and expected output data (results)).

The **Software Requirements Specification** (SRS-SW) describes all data, functional and non-functional requirements of the software under development, which is written from the user's perspective. This document has a legal character of business agreement with the customer.

- Validation/Verification method is embedded in the **Software Test Specification**, which describes the software test cases to be performed to validate that the system works according to SRS-SW. Software test cases describe what to test (functional and non-functional requirements, software interfaces and data sets, and quality attributes) and how to test (test scenarios, input test data, and expected output data (results)).

The **Software Architecture Specification** describes a component-oriented architecture of the software under development by focusing on key quality attributes, like reliability, performance, robustness, etc.

- Validation/Verification method is embedded in the **Integration Test Specification**, which describes the test cases to be performed to validate that the software architectural components interoperate with each other successfully. Integration test cases describe how to test full stake or partial integration of the software system. It describes also needed mocked, simulated and proxy components as replacements for those which are not developed or not interoperable yet.

The **Component/Unit Design Specification** provides detailed design for each software component including their interfaces.

- Validation/Verification method is embedded in **Component/Unit Test Specification**, which formally describes how to test the individual components/unit of a software. The unit test specification is itself an executable software module which relies on an automated unit testing framework, such as JUnit.

## 2.5 Supplementary Material

- Tutorials Group UNIT TESTING
  - Name: "Unit test with JUnit"
  - Locations:
    1. <https://www.vogella.com/tutorials/JUnit/article.html#:~:text=A%20unit%20test%20is%20a,a%20method%20or%20a%20class.>
    2. <https://www.guru99.com/junit-tutorial.html>
  - Name: "JUnit Testing in Eclipse"
  - Location: <https://www.youtube.com/watch?v=v2F49zLLj-8> (8'55)
- Course Video Clip
  - Name: "What is a Unit Test" ... with code example!
  - Location: <https://www.youtube.com/watch?v=4B4oKc7Cjdk> (4'01)
- Course Video Clip
  - Name: "Introduction to Scrum - 7 Minutes"
  - Location: <https://www.youtube.com/watch?v=9TycLR0TqFA>
- Course Video Clip Group AGILE
  - Name: "What is Agile?"
  - Location: <https://www.youtube.com/watch?v=Z9QbYZh1YXY&vl=de>

- Name: SAFe 5.0 Overview in Five Minutes
- Location: <https://www.youtube.com/watch?v=aW2m-BtCJyE>
- Name: "What is Unit Testing? - Software Testing Tutorial"
- Location: [https://youtu.be/lj5nnGa\\_Dlw](https://youtu.be/lj5nnGa_Dlw)

## 2.6 Exercises

1. Given is "*safety-critical industrial control software is tested differently from an e-commerce mobile app*". Recherche in internet and compose a 1-page discussion why and how they should be tested differently. Consider test objectives, test process, test tools and test people.
2. Given is "*testing in an Agile project is done differently than testing in a sequential software development lifecycle project*". Compose a 1-page comparison of both types of project focusing on objectives, process, tools and test people.
3. Show on which side of the V-Model the "Verification" and "Validation" are performed. Explain why.
4. What do you as a tester do if a requirement is not testable? Give such a requirement and propose 3 solution approaches, if possible, interview the testers in your company.

## 3 Concepts and Definitions

### 3.1 Fundamental Test Process

ISTQB 2018 Syllabus Topic Ch. 1.4 Test Process

A generic software test process comprises common sets of test activities, “without which testing will be less likely to achieve its established objectives”.

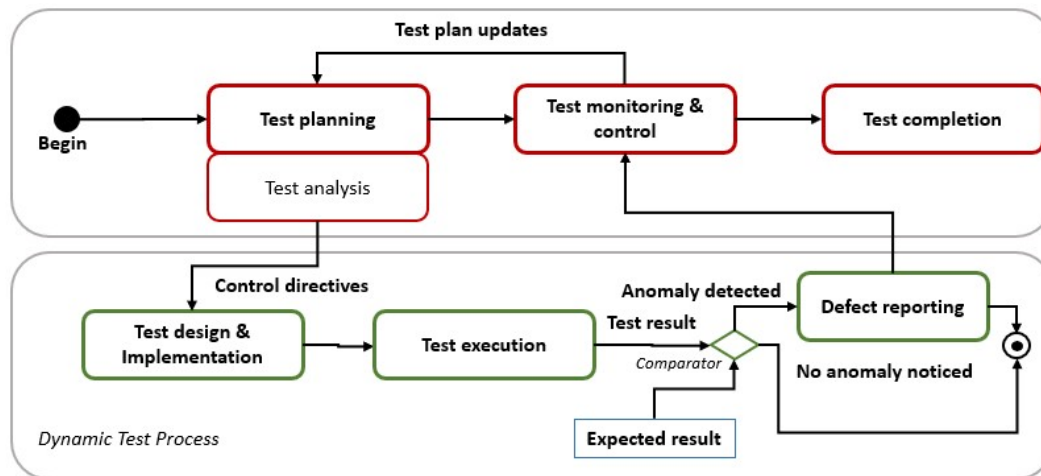


FIGURE 4 ISTQB FUNDAMENTAL TEST PROCESS

Dynamic Test Process: The activities “test design & implementation”, “test execution and reporting” in Figure 4 build the dynamic part of the process and thus they are performed in iterations to deliver a verified/validated increment of the software.

“A test process consists of the following main groups of activities: “

- **“Test planning:** involves activities that define the objectives of testing and the approach for meeting test objectives” according to a “test schedule for meeting a deadline.” Test plans are changed “based on feedback from monitoring and control activities.  
Test planning activities may include the following
  - Determining the scope, objectives, and risks of testing ...
  - Integrating and coordinating the test activities into the software lifecycle activities,
  - Making decisions about what to test, the people and [required] resources,
  - Scheduling of test analysis, design, implementation, execution, and evaluation activities, either on dates (e.g., in sequential development) or in the context of each iteration, (e.g., in iterative development)
  - Selecting metrics for test monitoring and control,
  - Budgeting for the test activities,
  - Determining the level of detail and structure for test documentation (e.g., by providing templates or example documents)
- **Test monitoring and control:** involves the on-going comparison of actual progress against planned progress using any test monitoring metrics defined in the test plan. Test control involves taking actions necessary to meet the objectives of the test plan (which may be updated over time). Test monitoring and control are supported by the evaluation of exit criteria, which are referred to as the definition of done in

some software development lifecycle models.

For example, the evaluation of exit criteria for test execution as part of a given test level may include:

- Checking test results and logs against specified coverage criteria,
- Assessing the level of component or system quality based on test results and logs,
- Determining if more tests are needed (e.g., if tests originally intended to achieve a certain level of product risk coverage failed to do so, requiring additional tests to be written and executed)
- **Test analysis:** determines “what to test” in terms of measurable coverage criteria.” Analogous to requirements analysis, the test basis is analyzed to determine what to validate and verify, and to define required product quality. “Test analysis includes the following major activities:
  - Analyzing the test basis” such as Requirement specifications, Design and implementation information, the implementation of the component or system itself, and Risk analysis reports, “which may consider functional, non-functional, and structural aspects of the component or system”
  - Evaluating the test basis<sup>4</sup> and test items to identify defects of various types, such as Ambiguities, Omissions, Inconsistencies, Inaccuracies, Contradictions, Superfluous statements,
  - Identifying features and sets of features to be tested,
  - Defining and prioritizing test conditions<sup>5</sup> for each feature,
  - Capturing bi-directional traceability between each element of the test basis and the associated test conditions.

The application of black-box, white-box, and experience-based test techniques can be useful in the process of test analysis to reduce the likelihood of omitting important test conditions and to define more precise and accurate test conditions.

- **Test design:** During test design, the test conditions are elaborated into high-level test cases .... So, test analysis answers the question “what to test?” while test design answers the question “how to test?”

Test design includes the following major activities:

- Designing and prioritizing test cases and sets of test cases
- Identifying necessary test data to support test conditions and test cases
- Designing the test environment and identifying any required infrastructure and tools
- Capturing bi-directional traceability between the test basis, test conditions, and test cases

The elaboration of test conditions into test cases and sets of test cases during test design often involves using test techniques (see “test technique black-box”).

- **Test implementation:** During test implementation,” test procedures including test cases are created. “Test design answers the question “how to test?” while test implementation answers the question “do we now have everything in place to run the tests?”

Test implementation includes the following major activities:

- Developing and prioritizing test procedures ...
- Creating test suites<sup>6</sup> from the test procedures ...
- Building the test environment (including, potentially, test harnesses<sup>7</sup>, service virtualization, simulators, and other infrastructure items) ...

---

<sup>4</sup> “Test basis: All documents from which the requirements of a component or system can be inferred. The documentation on which the test cases are based...”[ISTQBterms]

<sup>5</sup> “Test condition: An item or event of a component or system that could be verified by one or more test cases, e.g. a function, transaction, feature, quality attribute, or structural element.” [ISTQBterms]

<sup>6</sup> “Test suite: A set of several test cases for a component or system under test, where the post condition of one test is often used as the precondition for the next one. ”[ISTQBterms]

<sup>7</sup> “Test harness: A test environment comprised of stubs and drivers needed to execute a test.” [ISTQBterms]



- *Preparing test data ...*
- *Verifying and updating bi-directional traceability between the test basis, test conditions, test cases, test procedures, and test suites (see section “Traceability”)*

*Test design and test implementation tasks are often combined.*

- **Test execution:** *During test execution, test suites are run in accordance with the test execution schedule. Test execution includes the following major activities:*
  - *Recording the IDs and versions of the test item(s) or test object, test tool(s), and” test data*
  - *“Executing tests either manually or by using test execution tools*
  - *Comparing actual results with expected results” to detect anomalies*
  - *“Analyzing anomalies to establish their likely causes ...” e.g., defects*
  - *Reporting defects based on the failures observed ...*
  - *Logging the outcome of test execution (e.g., pass, fail, blocked) ...*
  - *Verifying and updating bi-directional traceability between the test basis, test conditions, test cases, test procedures, and test results.*
- **Test completion:** *Test completion activities collect data from completed test activities ... Test completion activities occur at project milestones such as when a software system is released, a test project is completed (or cancelled), an Agile project iteration is finished, a test level is completed, or a maintenance release has been completed.*

*Test completion includes the following major activities:*

- *Checking whether all defect reports are closed...*
- *Creating a test summary report to be communicated to stakeholders”,*
- *Archive the test environment, the test data, the test infrastructure, and other test relevant artefacts for later reuse,*
- *“Handing over” the test archive “to the maintenance teams ,*
- *Analyzing lessons learned from the completed test activities” to improve “test process maturity.”*

## 3.2 Test Work Products

*ISTQB 2018 Syllabus Topic Ch. 1.4.3 Test Work Products*

In general, a work product<sup>8</sup> is any kind of material which a worker has created and developed while working. In specific, a test work product<sup>9</sup> is any kind of material which a tester or test manager has created and developed during the test process phases. For example

- *“Test planning work products typically include one or more test plans”, where “the test plan includes information about the test basis, to which the other test work products will be related via traceability information ...*
- *Test monitoring and control work products typically include various types of test reports, including test progress reports produced ... on regular basis, and test summary reports produced at various completion milestones.*
- *Test analysis work products include defined and prioritized test conditions, each of which is ideally bidirectionally traceable to the specific element(s) of the test basis it covers. ...”*
- *Test design work products include “test cases and sets of test cases ...., test data, design of the test environment” and “the identification of infrastructure and tools.*

<sup>8</sup> Work product is the result of the activities of the corresponding test process phase!

<sup>9</sup> Test Work Products comprise a section in the Testing Document!

- Test implementation work products include test procedures and the sequencing of those test procedures, test suites and a test execution schedule...
- Test execution work products include the status of individual test cases e.g., ready to run, pass, fail, blocked, deliberately skipped, etc.
- Test completion work products include test summary reports..."

### 3.3 Traceability

ISTQB 2018 Syllabus Topic Ch. 1.4.4 Traceability between the Test Basis and Test Work Products

"In order to implement effective test monitoring and control, it is important to establish and maintain [bi-directional] traceability throughout the test process between each element of the test basis and the various test work products associated with that element.

Good traceability supports:

- Analyzing the impact of changes, ...
- Providing information to assess product quality, process capability, and project progress against business goals

As an example, the Figure 5 shows the downstream traceability of the system requirements and use cases.

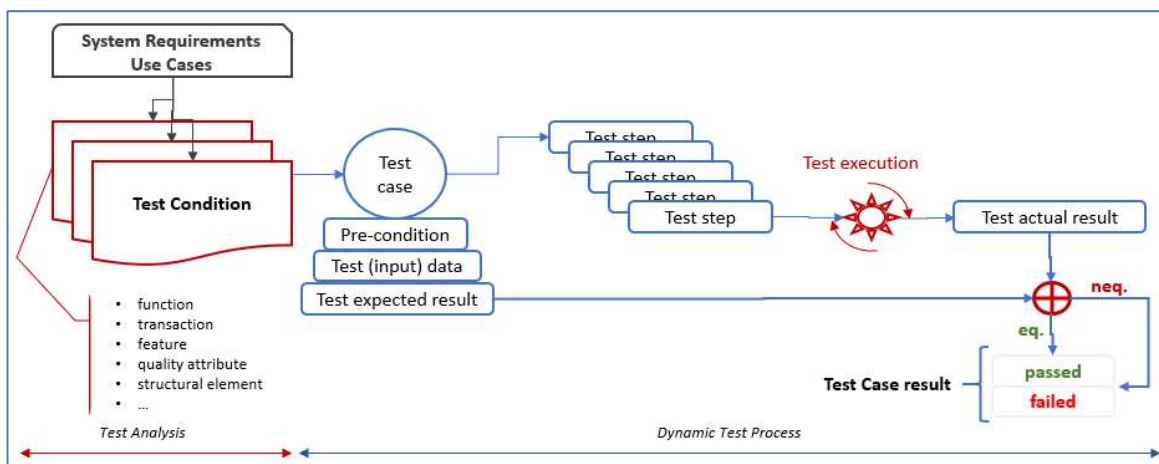


FIGURE 5 TRACEABILITY OF THE SYSTEM REQUIREMENTS AND USE CASES

### 3.4 Test Levels

ISTQB 2018 Syllabus Topic Ch. 2.2 Test Levels

"Test levels are groups of test activities that are organized and managed together.

The test levels used in this syllabus are:

- Component testing
- Integration testing
- System testing
- Acceptance testing

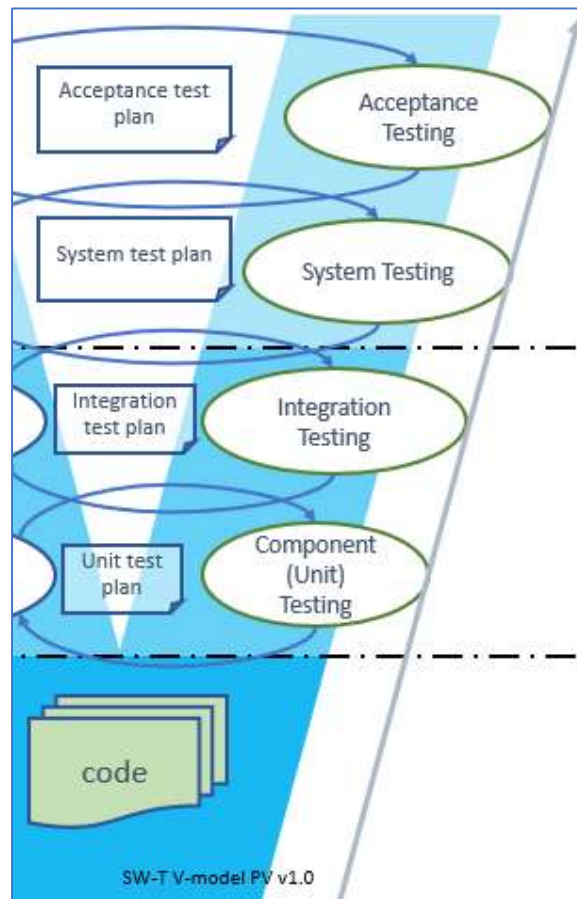


FIGURE 6 V-MODEL ILLUSTRATED FOR TEST LEVELS

“Each test level is an instance of the test process ... at a given level of [software] development. Test levels are related to other activities within the software development lifecycle”, for example, the horizontal validation/verification of software work products in V-Model.

“Test levels are characterized by the following attributes:

- Specific objectives
- Test basis, referenced to derive test cases
- Test object (i.e., what is being tested)
- Typical defects and failures
- Specific approaches and responsibilities”
- Appropriate test environment

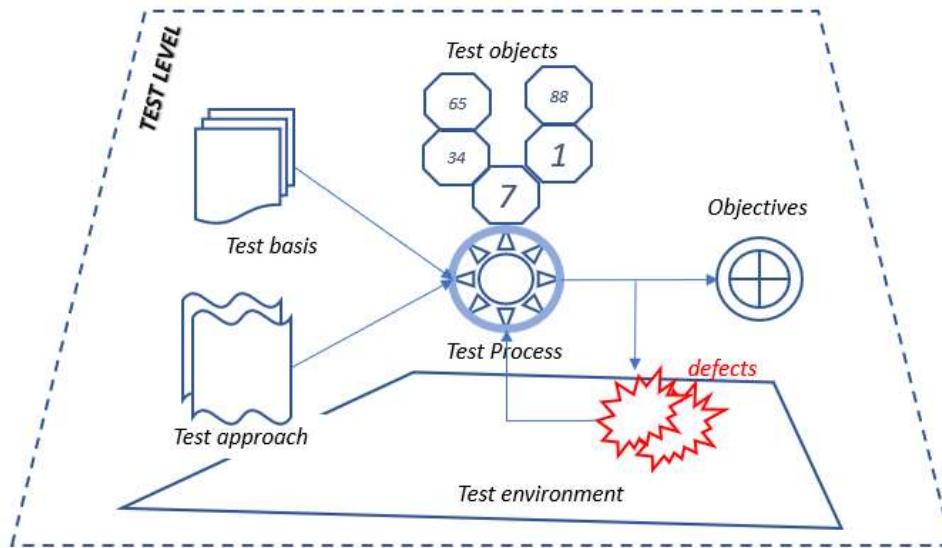


FIGURE 7 GENERIC TEST LEVEL



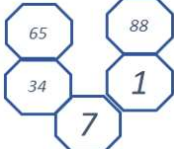


<b>Component testing</b>	
<b>"Objectives"</b> 	<ul style="list-style-type: none"> <li>• Reducing risk,</li> <li>• Verifying the functional and non-functional behaviors of the component</li> <li>• Building confidence in the component's quality</li> <li>• Finding defects in the component</li> <li>• Preventing defects from escaping to higher test levels</li> </ul>
<b>Test basis</b> 	<ul style="list-style-type: none"> <li>• Detailed design</li> <li>• Code</li> <li>• Data model</li> <li>• Component specifications</li> </ul>
<b>Test object</b> 	<ul style="list-style-type: none"> <li>• Components, units or modules</li> <li>• Code and data structures</li> <li>• Classes</li> <li>• Database modules</li> </ul>
<b>Typical defects and failures</b> 	<ul style="list-style-type: none"> <li>• Incorrect functionality (e.g., not as described in design specifications)</li> <li>• Data flow problems</li> <li>• Incorrect code and logic</li> </ul>
<b>Specific approaches and responsibilities</b> 	<ul style="list-style-type: none"> <li>• Component testing is performed by the developer</li> <li>• Developers write and execute tests after having written the code for a component."</li> <li>• Developers write and execute automated tests prior to written code (Agile development, Test-driven Development)</li> </ul>

TABLE 1 COMPONENT TESTING CHARACTERISTICS



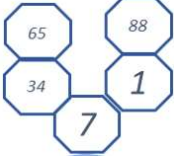


<b>Integration testing</b>	
<p><b>“Objectives</b></p> 	<ul style="list-style-type: none"> <li>• Reducing risk,</li> <li>• Verifying the functional and non-functional behaviors of the interfaces</li> <li>• Building confidence in the interfaces’ quality</li> <li>• Finding defects in the interfaces and components</li> <li>• Preventing defects from escaping to higher test levels</li> </ul>
<p><b>Test basis</b></p> 	<ul style="list-style-type: none"> <li>• Software and system design</li> <li>• Sequence diagrams</li> <li>• Interface and communication protocol specifications</li> <li>• Use cases</li> <li>• Architecture at component or system level</li> <li>• Workflows</li> <li>• External interface definitions</li> </ul>
<p><b>Test object</b></p> 	<ul style="list-style-type: none"> <li>• Subsystems</li> <li>• Databases</li> <li>• Infrastructure</li> <li>• Interfaces</li> <li>• APIs</li> <li>• Microservices</li> </ul>
<p><b>Typical defects and failures</b></p> 	<p><i>Component integration testing:</i></p> <ul style="list-style-type: none"> <li>• Incorrect data, missing data, or incorrect data encoding</li> <li>• Incorrect sequencing or timing of interface calls</li> <li>• Interface mismatch</li> <li>• Failures in communication between components</li> <li>• ... communication failures between components</li> </ul> <p><i>System integration testing:</i></p> <ul style="list-style-type: none"> <li>• Inconsistent message structures between systems</li> <li>• Incorrect data, missing data, or incorrect data encoding</li> <li>• Interface mismatch</li> <li>• Failures in communication between systems</li> <li>• ... communication failures between systems</li> </ul>
<p><b>Specific approaches and responsibilities</b></p> 	<ul style="list-style-type: none"> <li>• Component integration tests and system integration tests should concentrate on the integration itself.</li> <li>• Tests should focus on the communication between the modules” (Interoperability),” not the functionality of the individual modules</li> <li>• Component integration testing is often the responsibility of developers. System integration testing is generally the responsibility of [system] testers.</li> <li>• Integration should be incremental</li> <li>• A risk analysis of the most complex interfaces can help to focus the integration testing.</li> <li>• continuous integration, where software is integrated on a component-by-component basis (i.e., functional integration), has become common practice.</li> <li>• Continuous integration often includes automated regression testing, ideally at multiple test levels.”</li> </ul>

TABLE 2 INTEGRATION TESTING CHARACTERISTICS



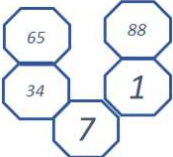



<b>System testing</b>	
<p><b>“Objectives</b></p> 	<ul style="list-style-type: none"> <li>• Reducing risk,</li> <li>• Verifying the functional and non-functional behaviors of the system</li> <li>• Validating that the system is complete and will work as expected</li> <li>• Building confidence in the quality of the system as a whole</li> <li>• Finding defects</li> <li>• Preventing defects from escaping to higher test levels or production</li> <li>• For certain systems, verifying data quality may also be an objective.</li> </ul>
<p><b>Test basis</b></p> 	<ul style="list-style-type: none"> <li>• System and software requirement specifications (functional and non-functional)</li> <li>• Risk analysis reports</li> <li>• Use cases</li> <li>• Epics and user stories</li> <li>• Models of system behavior</li> <li>• State diagrams</li> <li>• System and user manuals</li> </ul>
<p><b>Test object</b></p> 	<ul style="list-style-type: none"> <li>• Applications</li> <li>• Hardware/software systems</li> <li>• Operating systems</li> <li>• System under test (SUT)</li> <li>• System configuration and configuration data</li> </ul>
<p><b>Typical defects and failures</b></p> 	<ul style="list-style-type: none"> <li>• Incorrect calculations</li> <li>• Incorrect or unexpected system functional or non-functional behavior</li> <li>• Incorrect control and/or data flows within the system</li> <li>• Failure to properly and completely carry out end-to-end functional tasks</li> <li>• Failure of the system to work properly in the system environment(s)</li> <li>• Failure of the system to work as described in system and user manuals</li> </ul>
<p><b>Specific approaches and responsibilities</b></p> 	<ul style="list-style-type: none"> <li>• System testing should focus on the overall, end-to-end behavior of the system as a whole, both functional and non-functional.</li> <li>• System testing should use the most appropriate techniques”, like Black-box Testing</li> <li>• System testing is typically carried out by independent testers who rely heavily on specifications.</li> <li>• Early involvement of testers in user story refinement or static testing activities, such as reviews, helps to reduce the incidence” lists.</li> </ul>

TABLE 3 SYSTEM TESTING CHARACTERISTICS

<b>Acceptance Testing</b>	
<p><b>“Objectives</b></p> 	<ul style="list-style-type: none"> <li>• Establishing confidence in the quality of the system as a whole</li> <li>• Validating that the system is complete and will work as expected</li> <li>• Verifying that functional and non-functional behaviors of the system are as specified</li> <li>• Acceptance testing may also satisfy legal or regulatory requirements or standards.</li> <li>• Acceptance testing may produce information to assess the system’s readiness for deployment and use by the customer (end-user)</li> </ul>


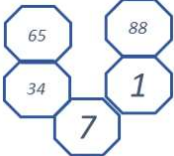


<p><b>Test basis</b></p> 	<ul style="list-style-type: none"> <li>• Business processes</li> <li>• User or business requirements</li> <li>• Regulations, legal contracts and standards</li> <li>• Use cases and/or user stories</li> <li>• System requirements</li> <li>• System or user documentation</li> <li>• Installation procedures</li> <li>• Risk analysis reports</li> </ul>
<p><b>Test object</b></p> 	<ul style="list-style-type: none"> <li>• System under test</li> <li>• System configuration and configuration data</li> <li>• Business processes for a fully integrated system</li> <li>• Recovery systems and hot sites (for business continuity and disaster recovery testing)</li> <li>• Operational and maintenance processes</li> <li>• Forms</li> <li>• Reports</li> <li>• Existing and converted production data</li> </ul>
<p><b>Typical defects and failures</b></p> 	<ul style="list-style-type: none"> <li>• Defects may be found during acceptance testing, but finding defects is not a goal in this testing level.</li> <li>• System workflows do not meet business or user requirements</li> <li>• Business rules are not implemented correctly</li> <li>• System does not satisfy contractual or regulatory requirements</li> <li>• Non-functional failures such as security vulnerabilities, inadequate performance efficiency under high loads, or improper operation on a supported platform</li> </ul>
<p><b>Specific approaches and responsibilities</b></p> 	<ul style="list-style-type: none"> <li>• Acceptance testing is often the responsibility of the customers, business users, product owners, or operators of a system, and other stakeholders may be involved as well.</li> </ul>

TABLE 4 ACCEPTANCE TESTING CHARACTERISTICS

### 3.5 Test Types

ISTQB 2018 Syllabus Topic Ch. 2.3 Test Types

“A test type is a group of test activities aimed at testing specific characteristics of a software system, or a part of a system, based on specific test objectives.” The characteristics of a software system is also known as the quality attributes of a software system.

The test types used in this syllabus are:

- Functional Testing
- Non-Functional Testing
- White-box Testing
- Change-related Testing

#### Functional Testing

“Functional testing of a system involves tests that evaluate functions that the system should perform.”

Functional testing is the testing of “HOW” the system behaves.

The test objective is to *“evaluate functional quality characteristics, such as completeness, correctness, and appropriateness of the software. Functional tests should be performed at all test levels (e.g., tests for components may be based on a component specification), though the focus is different at each level (see Ch. Test Level).”*

*Functional testing considers the behavior of the software, so black-box techniques (see Ch. “Black-box Techniques”) may be used to derive test conditions<sup>10</sup> and test cases<sup>11</sup> for the functionality.*

### **Non-functional Testing**

*Non-functional testing of a system evaluates characteristics of systems and software such as usability, performance efficiency or security.”*

Non-functional testing is the testing of “HOW WELL” the system behaves.

The test objective is to *“evaluate non-functional quality characteristics, such as reliability, performance, efficiency, security, compatibility, and usability. Non-functional testing can and often should be performed at all test levels and done as early as possible. The late discovery of non-functional defects can be extremely dangerous to the success of a project.*

*Black-box techniques<sup>12</sup> may be used to derive test conditions and test cases for nonfunctional testing.*

### **White-box Testing**

*White-box testing derives tests based on the system’s internal structure or implementation. Internal structure may include code, architecture, work flows, and/or data flows within the system.”*

The test objective is to *“evaluate whether the structure or architecture of the component or system is correct, complete, and as specified.*

*The thoroughness of white-box testing can be measured through structural coverage. At the component testing level, code coverage<sup>13</sup> is based on the percentage of component code that has been tested.”* At the component integration testing level, structural coverage is based on the percentage of component interfaces exercised by tests.

### **“Change-related Testing**

*When changes are made to a system, either to correct a defect or because of new or changing functionality, testing should be done to confirm that the changes have corrected the defect or implemented the functionality correctly.”*

The test objective is to *“evaluate the effects of changes, such as confirming that defects have been fixed (confirmation testing) and looking for unintended changes in behavior resulting from software or environment changes (regression testing).*

---

<sup>10</sup> *“Test condition: An item or event of a component or system that could be verified by one or more test cases, e.g., a function, transaction, feature, quality attribute, or structural element.” [ISTQBterm]*

<sup>11</sup> *“Test case : A set of input values, execution preconditions, expected results and execution postconditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement.” [ISTQBterm]*

<sup>12</sup> *“Black-box test design technique: Procedure to derive and/or select test cases based on an analysis of the specification, either functional or non-functional, of a component or system without reference to its internal structure.” [ISTQBterm]*

<sup>13</sup> *“Code coverage: An analysis method that determines which parts of the software have been executed (covered) by the test suite and which parts have not been executed, e.g., statement coverage, decision coverage or condition coverage.” [ISTQBterm]*



- **Confirmation testing:** *The purpose of a confirmation test is to confirm whether the original defect has been successfully fixed. After a defect is fixed, the software may be tested with all test cases that failed due to the defect [and] may also be tested with new tests to cover changes needed to fix the defect.*
- **Regression testing:** *Regression testing involves running tests to detect such unintended side-effects. Such unintended side-effects are called regressions. It is possible that a change made in one part of the code, whether a fix or another type of change, may accidentally affect the behavior of other parts of the code, whether within the same component, in other components of the same system, or even in other systems. Regression tests are run many times and generally evolve slowly, so regression testing is a strong candidate for automation.*

*Confirmation testing and regression testing are performed at all test levels.*

### 3.6 Supplementary Material

- Course Video Clip
  - Name: "Gear Inc. Software Testing Life Cycle"
  - Location: <https://www.youtube.com/watch?v=KW6F7aS9rSg> (3'42)
- Course Video Clip
  - Name: "How to write a TEST CASE? Software Testing Tutorial"
  - Location: <https://www.youtube.com/watch?v=BBmA5Qp6Ghk> (3'30)
- Course Video Clip
  - Name: "Acceptance Testing & System Testing - Software Testing Tutorial"
  - Location: <https://youtu.be/N8-qNMHOVyw>
- Course Video Clip
  - Name: "What is Integration Testing? Software Testing Tutorial"
  - Location: <https://youtu.be/QYCaaNz8emY>
- Course Video Clip
  - Name: "What is Regression Testing? Software Testing Tutorial"
  - Location: <https://youtu.be/aeu5zacsHsl>
- Course Video Clip
  - Name: "What is Test Scenario? How to write Test Scenario: Software Testing Tutorial"
  - Location: <https://youtu.be/wMN0pCyiQ9E>
- Course Video Clip
  - Name: "What is Test Basis ? Software Testing"
  - Location: <https://youtu.be/zz34cEC8Z4c>
- Course Video Clip
  - Name: "What is Manual Testing ?"
  - Location: <https://youtu.be/xCwkiZcEK6w>
- Course Video Clip
  - Name: "How Testing is Different in an Agile Project"
  - Location: [https://youtu.be/xdak981\\_v3g](https://youtu.be/xdak981_v3g)
- Course Video Clip Group "Test-driven Development"
  - Name: "JUnit 5 Basics 12 - Test driven development with JUnit" 5'00
  - Location: [https://youtu.be/zFJdQYn9u\\_8](https://youtu.be/zFJdQYn9u_8)
  - Name: "Test Driven Development Techniques - Part 1" 20'00
  - Location: <https://youtu.be/rQDlahWgOpk>
  - Name: "Test Driven Development Techniques - Part 2" 23'00

- Location: <https://youtu.be/AqFZ6mmCws8>
- Course Video Clip
  - Name: Requirement Traceability Matrix (RTM) in Software Testing
  - Location: <https://www.youtube.com/watch?v=cm-cSW66lsc&list=PLDC2A0C8D2EC934C7>

### 3.7 Exercises

1. If not already selected, select a concrete software system that you acquainted with, e.g. , a Social Media App, Bookstore App, a Shopping App, Embedded System App, etc. and use it as a context for the below exercises.
  - a) Extend the test level characteristics table of each test level with concrete examples.
  - b) Describe Testing in Agile Software Development
  - c) Describe Test-driven Development
2. Describe and illustrate in your own words the Component integration testing using other sources as ISTQB (2 pages with illustration)
3. Describe and illustrate in your own words the System integration testing using non-ISTQB sources. (2 pages with illustration)
4. Illustrate and compare in your own words the difference between Component Integration testing and System integration testing in ISTQB
5. Describe the benefits of planning the software integration testing using non-ISTQB sources.
6. Given is the following class to calculate a total price

```
import java.lang.Double;
import java.lang.Integer;

public class CalculatePrice {

    public Double calculateTotalPrice(Double basePrice,
                                     Double specialPrice,
                                     Double extraPrice,
                                     Integer extras,
                                     Double discount) {

        Double addonDiscount;
        Double result;

        if (extras >= 3) addonDiscount = 10;
        else if (extras >= 5) addonDiscount = 15;
        else addonDiscount = 0;

        if (discount > addonDiscount)
            addonDiscount = discount;
        result = basePrice/100.0*(100-discount)
            + specialPrice
            + extraPrice/100.0*(100-addonDiscount);
        return result;
    }
}
```

TABLE 5 CODE SAMPLE : CLASS CALCULATE

The class method contains a defect: The path “if (extras >= 5)” is not reachable. Such defects are detected by white box analysis

Write a test case to test this class in pseudocode! Explain in which Test Level you test the program and why!

7. *“The late discovery of non-functional defects can be extremely dangerous to the success of a project.”*

Argue why this statement is true especially for non-functional requirements of the system! You may use non-ISTQB sources!

8. *“It is possible to perform any of the test types mentioned above at any test level. ”*

For an application of your choice, give examples of functional, non-functional, white-box, and change-related tests for a test level of your choice. Use Ch. 2.3.5 in [ISTQB2018] as a reference.

## 4 Static Testing

ISTQB 2018 Syllabus Topic Ch. 3 Static Testing

The objective of static testing, just like the other testing activities, is to identify defects. The static testing succeeds particularly well this objective in the early phases in the software development process, even before a line of code is implemented. to reduce or eliminate costly changes and thus improve the software quality attributes.

### 4.1 Characteristics

*“In contrast to dynamic testing, which requires the execution of the software being tested, static testing relies on the manual examination of work products (i.e., reviews) or tool-driven evaluation of the code or other work products (i.e., static analysis). Both types of static testing assess the code or other work product being tested without actually executing the code or work product being tested.”*

Static analysis  
Review

*Almost any work product can be examined using static testing (reviews and/or static analysis), for example:”*

Work products under  
static testing

- Requirements specification
- “Epics, user stories, and acceptance criteria,
- Architecture and design specifications, [and] code”
- Test work products,
- User manuals,
- “Web pages,
- Contracts, project plans, schedules, and budget planning
- Configuration set up and infrastructure set up
- Models”, e.g., UML diagrams

### Differences between Static and Dynamic Testing

*“Static testing and dynamic testing can have the same objectives, such as providing an assessment of the quality of the work products and identifying defects as early as possible.*

*Static and dynamic testing complement each other by finding different types of defects.”*

The main distinctions are

- *“Static testing finds defects in work products directly rather than identifying failures caused by defects when the software is run. A defect can reside in a work product for a very long time without causing a failure”* because the defective code is never executed.
- *“Static testing can be used to improve the consistency and internal quality of work products, while dynamic testing typically focuses on externally visible behaviors.”*
- *“Compared with dynamic testing, ... defects ... are easier and cheaper to find and fix through static testing”* such as
  - Requirement defects
  - Design defects (e.g., inefficient algorithms or database structures, high coupling, low cohesion)
  - Coding defects (e.g., variables with undefined values, variables that are declared but never used, unreachable code, duplicate code)
  - Deviations from standards (e.g., lack of adherence to coding standards)
  - Incorrect interface specifications (e.g., different units of measurement used by the calling system than by the called system)

- *Security vulnerabilities (e.g., susceptibility to buffer overflows)*
- *Gaps or inaccuracies in test basis traceability or coverage (e.g., missing tests for an acceptance criterion)”*

The latter one, a.k.a. the requirements traceability review, is one of the most important static testing activities during the development of safety critical systems like airplane, cars, and trains.

### Reviews

*“Reviews vary from informal to formal. Informal reviews are characterized by not following a defined process and not having formal documented output. Formal reviews are characterized by team participation, documented results of the review, and documented procedures for conducting the review.*

*The formality of a review process is related to factors such as the software development lifecycle model, the maturity of the development process, the complexity of the work product to be reviewed, any legal or regulatory requirements, and/or the need for an audit trail.*

*The focus of a review depends on the agreed objectives of the review (e.g., finding defects, gaining understanding, educating participants such as testers and new team members, or discussing and deciding by consensus).”*

## 4.2 Supplementary Material

- Course Video Clip
  - Name: “What is Static Testing? What is a Review: Software Testing Tutorial 5’40
  - Location: <https://youtu.be/YosSY0e5Xy4>
- Course Video Clip
  - Name: “Code Collaborator v5.0 - Five Minute Demo” (CODE REVIEW TOOL) 5’00
  - Location: <https://youtu.be/R-nNuMuuLT4>
- Course Video Clip
  - Name: “Java Clean Code Tutorial #2 - Static Analysis FindBugs Eclipse Plugin Setup” (STATIC ANALYSIS TOOL for ECLIPSE) 5’00
  - Location: <https://youtu.be/lvNC6-k5xT8>

## 4.3 Exercises

*This section intentionally left blank!*

## 5 Test Techniques

ISTQB 2018 Syllabus Topic Ch. 4 Test Techniques

*“The purpose of a test technique is to help in identifying test conditions, test cases, and test data. The choice of which test techniques to use depends on a number of factors.”* Most commons are:

- *“Component or system complexity*
- *Risk levels and types*
- *Time and budget, [and]*
- *The types of defects expected in the component or system”*

In [ISTQB 2018] *“test techniques are classified as black-box, white-box, or experience-based”*. The first two techniques are described in the proceeding sections in detail.

*“Experience-based test techniques leverage the experience of developers, testers and users to design, implement, and execute tests. These techniques are often combined with black-box and white-box test techniques. Commonly used experience-based techniques are” “Error Guessing”, “Exploratory Testing” and “Checklist-based Testing”*. Application of one or all techniques in a software project depends on the project type, product type and maturity of experienced testers. A characteristic of this technique is that *“test conditions, test cases, and test data are derived from a test basis that may include knowledge and experience of testers, developers, users and other stakeholders.”*

### 5.1 Black-box Test Techniques

ISTQB 2018 Syllabus Topic Ch. 4.2 Black-box Test Techniques

*“Black-box test techniques (also called behavioral or behavior-based techniques) are based on an analysis of the appropriate test basis (e.g., formal requirements documents, specifications, use cases, user stories, or business processes). These techniques are applicable to both functional and nonfunctional testing. Black-box test techniques concentrate on the inputs and outputs of the test object without reference to its internal structure.*

Characteristics of this technique are

- *“Test conditions, test cases, and test data are derived from a test basis that may include software requirements, specifications, use cases, and user stories, [and]*
- *coverage is measured based on the items tested in the test basis and the technique applied to the test basis.”*

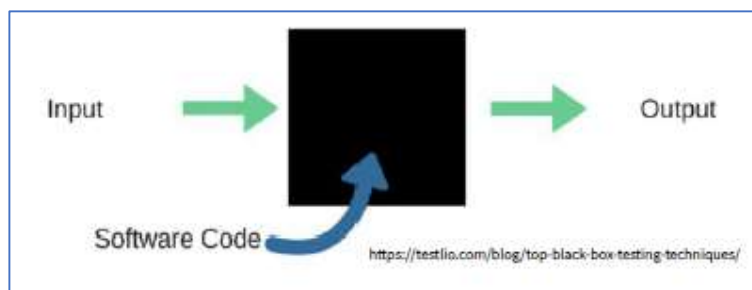


FIGURE 8 5.1 BLACK-BOX TEST

Some of the black-box techniques are described in the proceeding sections.

### 5.1.1 Equivalence Partitioning

“Equivalence partitioning divides data into partitions (also known as equivalence classes) in such a way that all the members of a given partition are expected to be processed in the same way.

There are equivalence partitions for both valid and invalid values.

Valid values are values that should be accepted by the component or system.

An equivalence partition containing valid values is called a “valid equivalence partition.”

Invalid values are values that should be rejected by the component or system.

An equivalence partition containing invalid values is called an “invalid equivalence partition.”

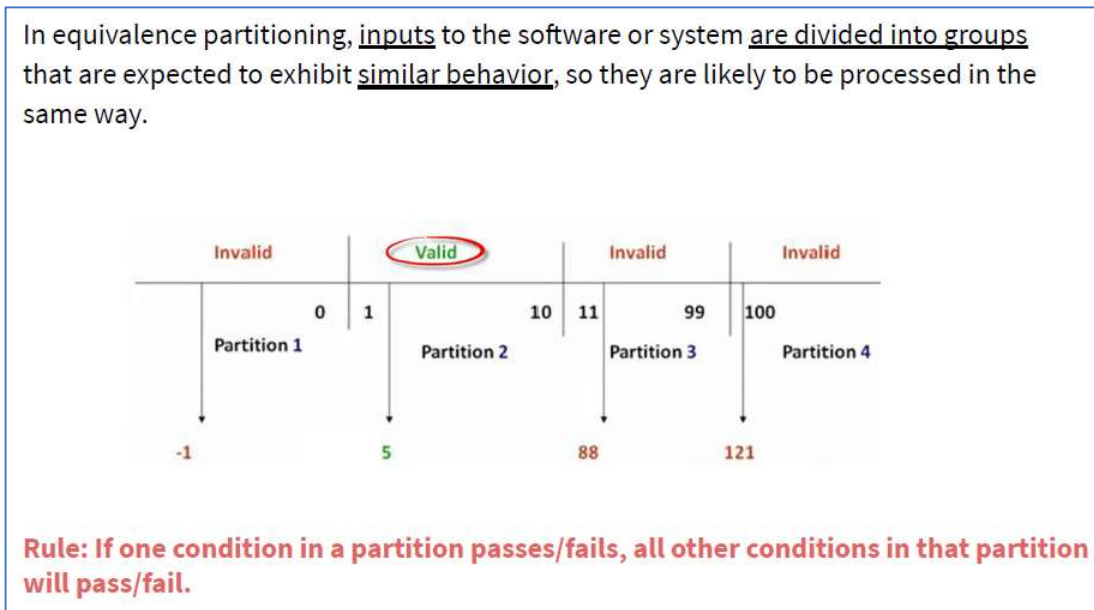


FIGURE 9 BLACK-BOX TEST TECHNIQUE - EQUIVALENCE PARTITIONING

Partitions can be identified for any data element related to the test object, including inputs, outputs, internal values, time-related values (e.g., before or after an event) and for interface parameters (e.g., integrated components being tested during integration testing).

Each [valid and invalid] value must belong to one and only one equivalence partition.

For a sales price of less than \$15,000, no discount shall be given. For a price up to \$20,000, a 5% discount is given. Below \$25,000, the discount is 7%, and from \$25,000 onward, the discount is 8.5%.”

Parameter	Equivalence classes	Representative
Sales price	vEC1: $0 \leq x < 15000$	14500
	vEC2: $15000 \leq x \leq 20000$	16500
	vEC3: $20000 < x < 25000$	24750
	vEC4: $x \geq 25000$	31800

Table 5-1  
Valid equivalence classes  
and representatives

FIGURE 10 EQUIVALENCE PARTITIONING - EXAMPLE

When invalid equivalence partitions are used in test cases, they should be tested individually, i.e., not combined with other invalid equivalence partitions, to ensure that failures are not masked.

To achieve 100% coverage with this technique, test cases must cover all identified partitions (including invalid partitions) by using a minimum of one value from each partition.

Coverage is measured as the number of equivalence partitions tested by at least one value, divided by the total number of identified equivalence partitions, normally expressed as a percentage.

Equivalence partitioning is applicable at all test levels.

## 5.1.2 Boundary Value Analysis

“Boundary value analysis (BVA) is an extension of equivalence partitioning but can only be used when the partition is ordered, consisting of numeric or sequential data. The minimum and maximum values (or first and last values) of a partition are its boundary values.

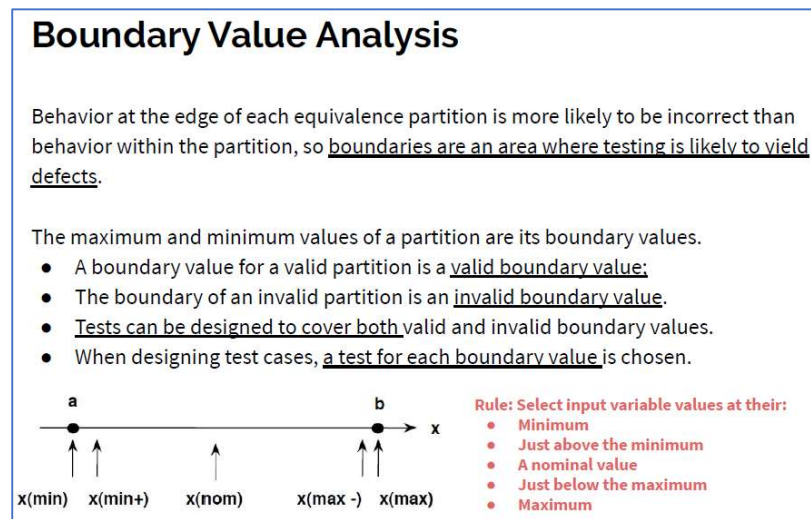


FIGURE 11 BLACK-BOX TEST TECHNIQUE - BOUNDARY VALUE ANALYSIS

For example, suppose an input field accepts a single integer value as an input, using a keypad to limit inputs so that non-integer inputs are impossible.

The valid range is from 1 to 5, inclusive.

So, there are three equivalence partitions:

- invalid (too low);
- valid;
- invalid (too high).

For the valid equivalence partition, the boundary values are 1 and 5.

For the invalid (too high) partition, the boundary value is 6.

For the invalid (too low) partition, there is only one boundary value, 0, because this is a partition with only one member.

Example

In the example above, we identify two boundary values per boundary. The boundary between invalid (too low) and valid gives the test values 0 and 1. The boundary between valid and invalid (too high) gives the test values 5 and 6.



Some variations of this technique identify three boundary values per boundary: the values before, at, and just over the boundary. In the previous example, using three-point boundary values, the lower boundary test values are 0, 1, and 2, and the upper boundary test values are 4, 5, and 6.

Behavior at the boundaries of equivalence partitions is more likely to be incorrect than behavior within the partitions. “

“It is important to remember that both specified and implemented boundaries may be displaced to positions above or below their intended positions, may be omitted altogether, or may be supplemented with unwanted additional boundaries. Boundary value analysis and testing will reveal almost all such defects by forcing the software to show behaviors from a partition other than the one to which the boundary value should belong.”

Benefit

Boundary value analysis can be applied at all test levels.

This technique is generally used to test requirements that call for a range of numbers (including dates and times).

Boundary coverage for a partition is measured as the number of boundary values tested, divided by the total number of identified boundary test values, normally expressed as a percentage.

The table below shows the different conditions and the truth values of the corresponding boundary values.

**Table 5-8**  
Table with three boundary values to test the condition

Implemented condition	24999	25000	25001	Remark
X < 25000 (correct)	True	False	False	Expected result
X ≤ 25000	True	True	False	25000 finds the fault
X <> 25000	True	False	True	25001 find the fault
X > 25000	False	False	True	24999 and 25001 find the fault
X ≥ 25000	False	True	True	All three values find the fault
X == 25000	False	True	False	24999 and 25000 find the fault

FIGURE 12 BOUNDARY VALUE ANALYSIS - EXAMPLE

### 5.1.3 State Transition Testing

“Components or systems may respond differently to an event depending on current conditions ... A state transition diagram shows the possible software states, as well as how the software enters, exits, and transitions between states.

A transition is initiated by an event (e.g., user input of a value into a field). The event results in a transition. The same event can result in two or more different transitions from the same state.

The state change may result in the software taking an action (e.g., outputting a calculation or error message).

*A state transition table shows all valid transitions and potentially invalid transitions between states, as well as the events, and resulting actions for valid transitions.*

## **State Transition Testing**

A system may exhibit a different response depending on current conditions or previous history (its state).

In this case, that aspect of the system can be shown with a state transition diagram.

A state table shows the relationship between the states and inputs, and can highlight possible transitions that are invalid.

Tests can be designed to cover a typical sequence of states, to cover every state, to exercise every transition, to exercise specific sequences of transitions or to test invalid transitions.

**FIGURE 13 BLACK-BOX TEST TECHNIQUE - STATE TRANSITION TESTING**

*State transition diagrams normally show only the valid transitions and exclude the invalid transitions.*

*Tests can be designed to cover a typical sequence of states, to exercise all states, to exercise every transition, to exercise specific sequences of transitions, or to test invalid transitions.*

*State transition testing is used for menu-based applications and is widely used within the embedded software industry.*

*The concept of a state is abstract – it may represent a few lines of code or an entire business process.*

*Coverage is commonly measured as the number of identified states or transitions tested, divided by the total number of identified states or transitions in the test object, normally expressed as a percentage.*

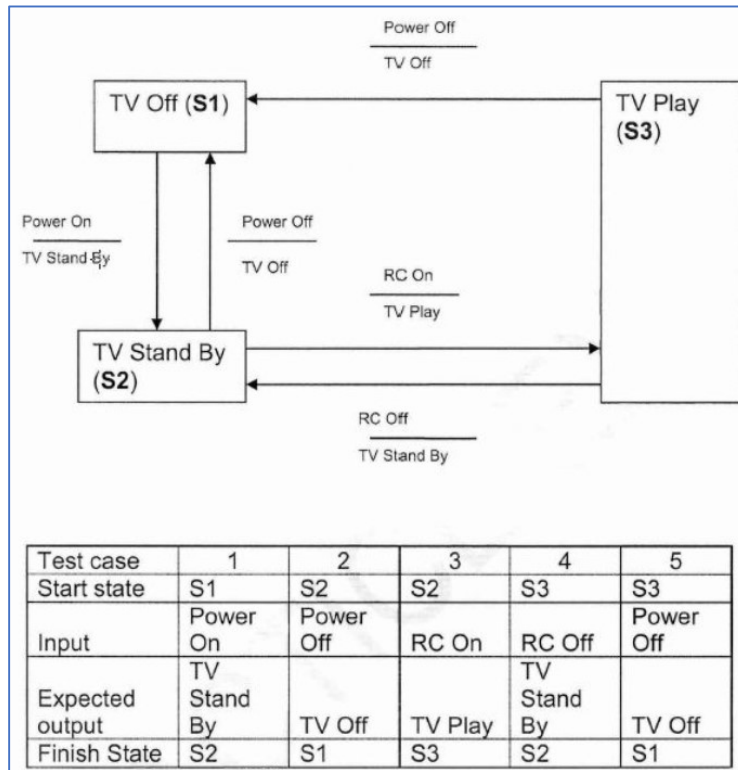


FIGURE 14 STATE TRANSITION TESTING - EXAMPLE

### 5.1.4 Use Case Testing

Tests can be derived from use cases, which are a specific way of designing interactions with software items. They incorporate requirements for the software functions.

Use cases are associated with actors (human users, external hardware, or other components or systems) and subjects (the component or system to which the use case is applied).

Each use case specifies some behavior that a subject can perform in collaboration with one or more actors<sup>14</sup> (authors note).

A use case can be described by interactions and activities, as well as preconditions, postconditions and natural language where appropriate. Interactions between the actors and the subject may result in changes to the state of the subject. Interactions may be represented graphically by work flows, activity diagrams, or business process models.

<sup>14</sup> "User or any other person or system that interacts with the system under test in a specific way." [ISTQBterm]

## Use Case Testing

Use Cases are in general provided functionality described from the viewpoint of a user

- at the business process level (business use cases) or
- at the system level (system use cases).

A use case describes interactions between actors (users or systems), which produce a result of value to a system user or the customer.

Use cases describe the “process flows” through a system based on its actual likely use, so the test cases derived from use cases are most useful in uncovering defects in the process flows during real-world use of the system.

Tests can be derived from use cases.

FIGURE 15 BLACK-BOX TEST TECHNIQUE - USE CASE TESTING

*A use case can include possible variations of its basic behavior, including exceptional behavior and error handling (system response and recovery from programming, application and communication errors, e.g., resulting in an error message). Tests are designed to exercise the defined behaviors (basic, exceptional or alternative, and error handling).*

Use Case: ATM - Card Validation		
	Step	Description
<b>Main Success Scenario</b> A: Actor S: System	1	A: Inserts card
	2	S: Validates card and asks for PIN
	3	A: Enters PIN
	4	S: Validates PIN
	5	S: Allows access to account
<b>Extensions</b>	2a	Card not valid S: Display message and reject card
	4a	PIN not valid S: Display message and ask for re-try (twice)
	4b	PIN invalid 3 times S: Eat card and exit

FIGURE 16 USE CASE TESTING

*Coverage can be measured by the number of use case behaviors tested divided by the total number of use case behaviors, normally expressed as a percentage.*

## 5.2 White-box Test Techniques

ISTQB 2018 Syllabus Topic Ch. 4.3 White-box Test Techniques

“White-box test techniques (also called structural or structure-based techniques) are based on an analysis of the architecture, detailed design, internal structure, or the code of the test object. Unlike black-box test techniques, white-box test techniques concentrate on the structure and processing within the test object.”

Characteristics of this technique are “

- Test conditions, test cases, and test data are derived from a test basis that may include code, software architecture, detailed design, or any other source of information regarding the structure of the software, [and]
- coverage is measured based on the items tested within a selected structure (e.g., the code or interfaces) and the technique applied to the test basis.”

“White-box testing is based on the internal structure of the test object. White-box test techniques can be used at all test levels, but the two code-related techniques discussed in this section are most commonly used at the component test level.”

### Statement Testing and Coverage

Statement testing exercises the potential executable statements in the code.

Coverage is measured as the number of statements executed by the tests divided by the total number of executable statements in the test object normally expressed as a percentage.

### Statement Testing and Coverage

The statement testing technique derives test cases to execute specific statements, normally to increase code coverage.

Statement coverage is determined by

- (the number of executed statements by a test case suite)

$$\frac{\text{the number of executed statements by a test case suite}}{\text{the number of all executable statements}} = \dots [\%]$$

FIGURE 17 WHITE-BOX TEST TECHNIQUE - STATEMENT TESTING AND COVERAGE

### Statement Coverage Method

- Count all the linearly independent paths
- Pick the minimum number of linearly independent paths that will include all the statements (S's and C's in the diagram)

Path1 : S1 - C1 - S3  
 Path2 : S1 - C1 - S2 - S3

**Path1 and Path2 are needed to cover all the statements: (S1,C1,S2,S3) ?**

```

        graph TD
            S1[S1] -- 1 --> C1{C1}
            C1 -- 2 --> S2[S2]
            C1 -- 3 --> S3[S3]
            S2 -- 3 --> S3
            S3 -- 4 --> S1
        
```

FIGURE 18 STATEMENT COVERAGE METHOD

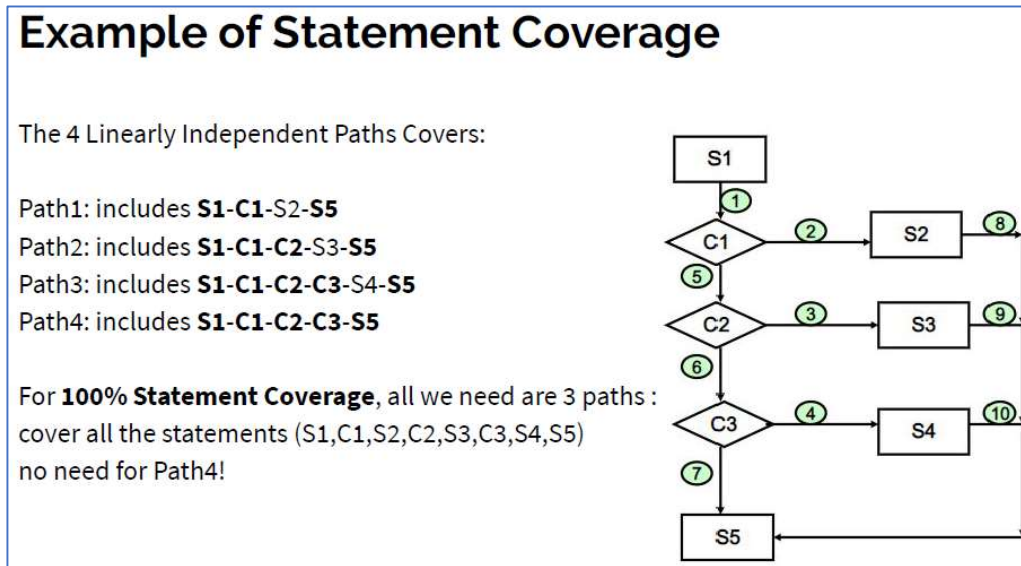


FIGURE 19 STATEMENT COVERAGE EXAMPLE

**Decision Testing and Coverage**

Decision testing exercises the decisions in the code and tests the code that is executed based on the decision outcomes. To do this, the test cases follow the control flows that occur from a decision point (e.g., for an IF statement, one for the true outcome and one for the false outcome; for a CASE statement, test cases would be required for all the possible outcomes, including the default outcome).

Coverage is measured as the number of decision outcomes executed by the tests divided by the total number of decision outcomes in the test object, normally expressed as a percentage.

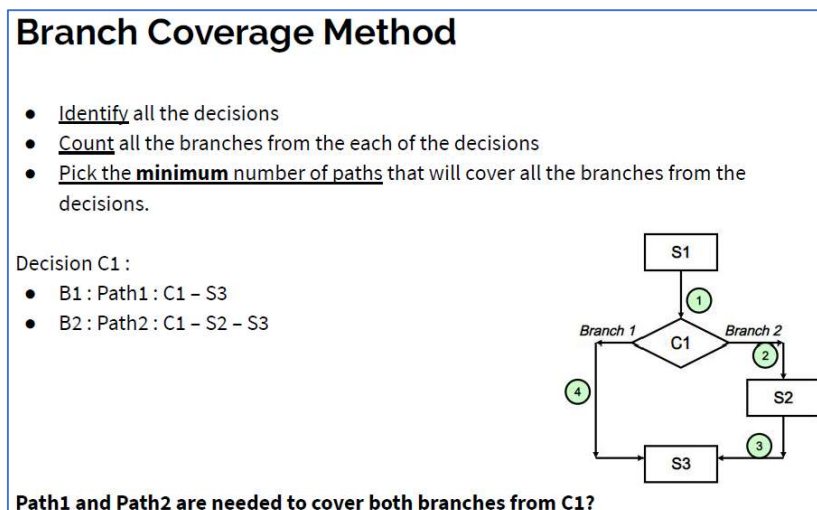


FIGURE 20 BRANCH COVERAGE METHOD

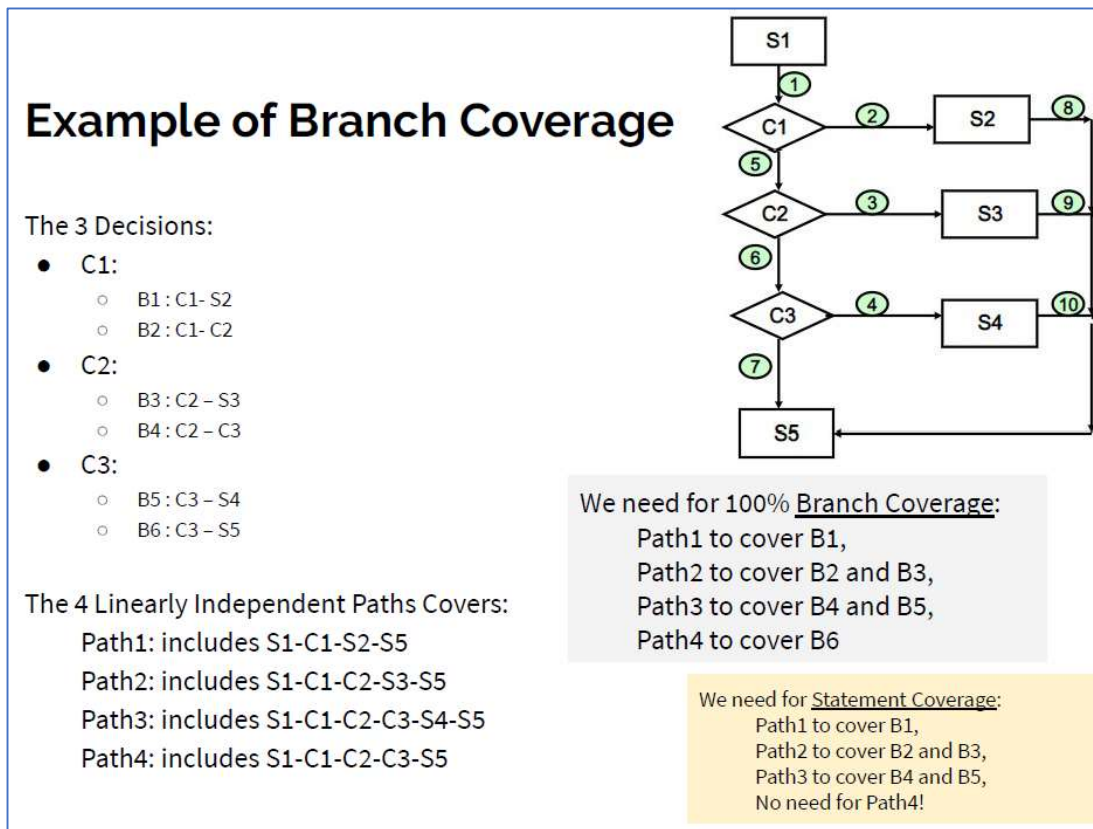


FIGURE 21 BRANCH COVERAGE EXAMPLE

### The Value of Statement and Decision Testing

When 100% statement coverage is achieved, it ensures that all executable statements in the code have been tested at least once, but it does not ensure that all decision logic has been tested. Of the two white-box techniques [above], statement testing may provide less coverage than decision testing.

When 100% decision coverage is achieved, it executes all decision outcomes, which includes testing the true outcome and also the false outcome, even when there is no explicit false statement (e.g., in the case of an IF statement without an else in the code).

Statement coverage helps to find defects in code that was not exercised by other tests.

Decision coverage helps to find defects in code where other tests have not taken both true and false outcomes.

Achieving 100% decision coverage guarantees 100% statement coverage (but not vice versa).

## 5.3 Supplementary Material

- Course Video Clip Group “Black-box Test Techniques”
  - Name: Boundary Value Analysis and Equivalence Partitioning: Software Testing Tutorial
  - Location: <https://www.youtube.com/watch?v=P1Hv2sUPKeM&list=PLDC2A0C8D2EC934C7&index=15>
  - Name: State Transition Testing: Software Testing Tutorial 17

- Location:  
<https://www.youtube.com/watch?v=Udjai6b9Y&list=PLDC2A0C8D2EC934C7&index=17>
- Name: *Use Case Testing: Software Testing Tutorial 18*
- Location:  
<https://www.youtube.com/watch?v=ijtvAvapsP0&list=PLDC2A0C8D2EC934C7&index=18>
  
- Course Video Clip Group “White-box Test Techniques”
  - Name: *What is White Box Testing? Tutorial with Examples*
  - Location:  
<https://www.youtube.com/watch?v=3bJcvBLjViQ&list=RDCMUC19i1XD6k88KqHIET8atqFQ&index=4>

## 5.4 Exercises

*This section intentionally left blank!*



## 6 Test Tools

ISTQB 2018 Syllabus Topic Ch. 6 Tool Support for Testing

### 6.1 Test Tools Basics

#### Supported Activities

*“Test tools can be used to support one or more testing activities. Such tools include:*

- *Tools that are used*
  - *directly in testing, such as test execution tools and test data preparation tools*
- *Tools that help to manage*
  - *requirements, test cases, test procedures, automated test scripts, test results, test data, and defects, and for reporting and monitoring test execution*
- *Tools that are used*
  - *for analysis and evaluation.*
- *Any tool that assists*
  - *in testing (a spreadsheet is also a test tool in this meaning)*

#### Tool Classification

*[Test] tools are classified according to the test activities that they support:*

- *Improve the efficiency of test activities by automating repetitive tasks or tasks that require significant resources when done manually (e.g., test execution, regression testing)*
- *Improve the efficiency of test activities by supporting manual test activities throughout the test process*
- *Improve the quality of test activities by allowing for more consistent testing and a higher level of defect reproducibility*
- *Automate activities that cannot be executed manually (e.g., large scale performance testing)*
- *Increase reliability of testing (e.g., by automating large data comparisons or simulating behavior)*

*In order to have a smooth and successful implementation, there are a number of things that ought to be considered when selecting and integrating test execution and test management tools into an organization.*

#### Test execution tools

*Test execution tools execute test objects using automated test scripts. This type of tools often requires significant effort in order to achieve significant benefits.*

- *Capturing test approach:*  
*Capturing tests by recording the actions of a manual tester seems attractive, but this approach does not scale to large numbers of test scripts. A captured script is a linear representation with specific data and actions as part of each script. This type of script may be unstable when unexpected events occur and require ongoing maintenance as the system’s user interface evolves over time.*
- *Data-driven test approach:*  
*This test approach separates out the test inputs and expected results, usually into a spreadsheet, and uses a more generic test script that can read the input data and execute the same test script with different data.*
- *Keyword-driven test approach:*  
*This test approach, a generic script processes keywords describing the actions to be taken (also called action words), which then calls keyword scripts to process the associated test data.*

*Regardless of the scripting technique used, the expected results for each test need to be compared to actual results from the test, either dynamically (while the test is running) or stored for later (post-execution) comparison.*

### **Test management tools**

*Test management tools often need to interface with other tools or spreadsheets for various reasons, including:*

- *To produce useful information in a format that fits the needs of the organization*
- *To maintain consistent traceability to requirements in a requirements management tool*
- *To link with test object version information in the configuration management tool*

*This is particularly important to consider when using an integrated tool (e.g., Application Lifecycle Management), which includes a test management module, as well as other modules (e.g., project schedule and budget information) that are used by different groups within an organization.”*

## 6.2 Supplementary Material

- **Tutorials**
  - Name: “Java Code Coverage for Eclipse”
  - Location: <https://www.elemma.org/index.html>

## 6.3 Exercises

1. Find and evaluate a test tool that supports “Capturing test approach”. Document the result of your evaluation in 3 – 5 MS PowerPoint slides.
2. Find and evaluate a test tool that supports “Data-driven test approach”. Document the result of your evaluation in 3 – 5 MS PowerPoint slides.
3. Find and evaluate a test tool that supports “Keyword-driven test approach”. Document the result of your evaluation in 3 – 5 MS PowerPoint slides.

## 7 Risks and Testing

ISTQB 2018 Syllabus Topic Ch. 5.5 Risks and Testing

### 7.1 Risk Definition

“Risk involves the possibility of an event in the future which has negative consequences. The level of risk is determined by the likelihood of the event and the impact (the harm) from that event.

		Impact			
		Minor	Moderate	Major	Extreme
Probability	Rare	Low	Low	Medium	Medium
	Unlikely	Low	Medium	Medium	Medium
	Moderate	Medium	Medium	Medium	High
	Likely	Medium	Medium	High	High
	Very likely	Medium	High	High	High

FIGURE 22 LEVELS OF RISKS IN MATRIX FORM [HSSE]

**Risk matrix to measure the level of risk:**  
**Risk (rating) = LIKELIHOOD x IMPACT**

Side note: Probability is a synonym for likelihood, and consequence (or severity) for impact.

The risk matrix is domain-agnostic and may be used in any domain to determine the level of risk, but the levels of probability and impact should be discussed for each risk to determine the appropriate actions to mitigate the risk.

In the following sections, the project and product risks are discussed in the software testing domain.

### 7.2 Product Risks

“Product risk involves the possibility that a work product (e.g., a specification, component, system, or test) may fail to satisfy the legitimate needs of its users and/or stakeholders.

When the product risks are associated with specific quality characteristics of a product (e.g., functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability, and portability), product risks are also called quality risks.

Product quality

Examples of product risks include:

- Software might not perform its intended functions according to the specification
- Software might not perform its intended functions according to user, customer, and/or stakeholder needs
- A system architecture may not adequately support some non-functional requirement(s)
- A particular computation may be performed incorrectly in some circumstances
- A loop control structure may be coded incorrectly
- Response-times may be inadequate for a high-performance transaction processing system
- User experience (UX) feedback might not meet product expectations”

Risk examples

### 7.3 Project Risks

“Project risk involves situations that, should they occur, may have a negative effect on a project's ability to achieve its objectives.

Project objectives

*Examples of project risks include:*

- *Delays may occur in delivery, task completion, or satisfaction of exit criteria or definition of done*
- *Inaccurate estimates, reallocation of funds to higher priority projects, or general cost-cutting across the organization may result in inadequate funding*
- *Late changes may result in substantial re-work*
  
- *Skills, training, and staff may not be sufficient*
- *Personnel issues may cause conflict and problems*
- *Users, business staff, or subject matter experts may not be available due to conflicting business priorities*
  
- *Testers may not communicate their needs and/or the test results adequately*
- *Developers and/or testers may fail to follow up on information found in testing and reviews (e.g., not improving development and testing practices)*
- *There may be an improper attitude toward, or expectations of, testing (e.g., not appreciating the value of finding defects during testing)*
  
- *Requirements may not be defined well enough*
- *The requirements may not be met, given existing constraints*
- *The test environment may not be ready on time*
- *Data conversion, migration planning, and their tool support may be late*
- *Weaknesses in the development process may impact the consistency or quality of project work products such as design, code, configuration, test data, and test cases*
- *Poor defect management and similar problems may result in accumulated defects and other technical debt*
  
- *A third party may fail to deliver a necessary product or service, or go bankrupt*
- *Contractual issues may cause problems to the project"*

*Risk examples*

Risky project issues

Risky organizational issues

Risky political issues

Risky technical issues

Risky supplier issues

## 7.4 Risk-based Testing and Product Quality

*"Risk is used to decide where and when to start testing and to identify areas that need more attention.*

*Testing is used to reduce the probability of an adverse event occurring, or to reduce the impact of an adverse event.*

*Testing is used as a risk mitigation activity, to provide information about identified risks, as well as providing information on residual (unresolved) risks.*

*A risk-based approach to testing provides proactive opportunities to reduce the levels of product risk. It involves product risk analysis, which includes the identification of product risks and the assessment of each risk's likelihood and impact. The resulting product risk information is used to guide test planning, the specification, preparation and execution of test cases, and test monitoring and control.*

*Risk-based approach*

*Analyzing product risks early contributes to the success of a project.*

In a risk-based approach, the results of product risk analysis are used to:

- Determine the test techniques to be employed
- Determine the particular levels and types of testing to be performed (e.g., security testing, accessibility testing)
- Determine the extent of testing to be carried out
- Prioritize testing in an attempt to find the critical defects as early as possible
- Determine whether any activities in addition to testing could be employed to reduce risk (e.g., providing training to inexperienced designers)”

### 7.5 Risk Management procedure

Risk management is imperative and one of the most important pillars in software development projects. Therefore, it should be institutionalized and performed continuously. Additionally, identified and rated risks should be considered in the dawn of every critical decision.

“Risk management activities provide a disciplined approach to:

1. Analyze (and re-evaluate on a regular basis) what can go wrong (risks)
2. Determine which risks are important to deal with
3. Implement actions to mitigate those risks
4. Make contingency [probability] plans to deal with the risks should they become actual events [impacts]”
5. In addition, testing may identify new risks, help to determine what risks should be mitigated, and lower uncertainty about risks.”

Risk Management procedure

Analysis

Identify and prioritize

Mitigation action development

Proactive risk monitoring

Testing’s contribution

		Impact			
		Minor	Moderate	Major	Extreme
Probability	Rare	Low	Low	Medium	Medium
	Unlikely	Low	Medium	Medium	Medium
	Moderate	Medium	Medium	Medium	High
	Likely	Medium	Medium	High	High
	Very likely	Medium	High	High	High

FIGURE 23 MITIGATION ACTION DEVELOPMENT

## 7.6 Exercises

1. Apply the steps 1 and 2 of the risk management procedure to your selected system to identify 5 requirements. Select 2 risky requirements and determine the risk level on the risk matrix. Define 1 mitigation action for each to reduce its risk level.
2. In section “Project Risks” the column contains 5 type of project issues. From each type select a risk and discuss how to mitigate it.

## 8 References

- [HSSE] ... Risk Matrix figure <https://hsseworld.com/assessing-risks/>
- [ISTQB2018] ... the ISTQB Certified Tester Foundation Level Syllabus Version 2018 V3.1
- [ISTQBterms] ... Standard Glossary of Terms Used in Software Testing, Version 3.01, All Terms, ISTQB
- [STFebook] ...  
[http://prof.mau.ac.ir/images/Uploaded\\_files/Software%20Testing%20Foundations%20A%20Study%20Guide%20for%20the%20Certified%20Tester%20Exam%5B5309302%5D.PDF](http://prof.mau.ac.ir/images/Uploaded_files/Software%20Testing%20Foundations%20A%20Study%20Guide%20for%20the%20Certified%20Tester%20Exam%5B5309302%5D.PDF)

## 9 ANNEX – TOPIC SECTION

*ISTQB 2018 Syllabus Topic Ch. <# - name>*

[intro if sub-topics present]

### 9.1 Overview or 1st sub-topic if given

< essentials>, < Illustrations>, <citations>

### 9.2 2nd sub-topic if given

< essentials>, < Illustrations>, <citations>

### 9.3 Exercises

<at least mocked questions & answers>

### 9.4 Supplementary Material

- **Slides (complementary [Group <NAME>])**
  - Name and Location:
- **Explanatory Illustrations [Group <NAME>]**
  - Name and Location:
- **Explanatory Video Clips [Group <NAME>]**
  - Name and Location:
- **Tutorials [Group <NAME>]**
  - Name and Location:
- **Templates and Specifications [Group <NAME>]**
  - Name and Location:
- **Code and Tooling [Group <NAME>]**
  - Name and Location:
- **Course Video Clip [Group <NAME>]**
  - Name and Location:
  - ...
- **Exams [Group <NAME>]**
  - Name and Location: