ISTQB - Certified Tester Foundation Level (CTFL) v4.0

1. Fundamentals of Testing

What is Testing?
Software testing is a set of activities to discover defects and evaluate the quality of software work products.

Why is Testing Necessary?
Testing, as a form of quality control, helps in achieving the agreed upon test objectives within the set scope, time, quality, and budget constraints.

Testing and Debugging
- **Testing** can trigger failures that are caused by defects in the software (dynamic testing) or can directly find defects in the test object (static testing)
- **Debugging** is concerned with finding causes of this failure (defects), analyzing and eliminating them.

Testing and Quality Assurance
Testing is performed as a part of QC.

- Testing is product-oriented
- QA is process-oriented

Errors, Defects, Failues, and Root Causes
Human beings make errors (mistakes), which produce defects (faults, bugs), which may result in failures. Failures can also be caused by environmental conditions.

Testing Principles
1. Testing shows the presence of defects.
2. Exhaustive testing is impossible.
3. Early testing saves time and money.
4. Defects cluster together.
5. Tests wear out.
6. Testing is context dependent.
7. Absence-of-defects fallacy.

Test Activities and Tasks
- Test planning
- Test monitoring and test control
- Test analysis
- Test design
- Test implementation
- Test execution
- Test completion

Roles in Testing
- **Test Management Role:** Takes overall responsibility for the test process, test team and leadership of the test activities.
- **Testing Role:** Focused on the activities of test analysis, test design, test implementation and execution.

Good Practices
- **Whole Team Approach:** Any team member can perform any task, everyone is responsible for quality.
- **Independence of Testing:** Independent testers recognize different kinds of failures and defects, but may be isolated from the development team, causing poor collaboration.

2. Testing Throughout the SDLC

Software Development Lifecycle (SDLC)
An SDLC model is an abstract, high-level representation of the software development process. Every software development activity has a corresponding test activity.

Testing as a Driver for Software Development
- **Test-Driven Development (TDD)**
  Tests are written first, then the code is written to satisfy the tests.
- **Acceptance Test-Driven Development (ATDD)**
  Tests are derived from acceptance criteria and written before development of the application part.
- **Behavior-Driven Development (BDD)**
  Tests are written in simple natural language (e.g., Given/When/Then) to express desired behavior.

Shift Left
Approach where testing is performed earlier.

Test Levels
Test levels are groups of test activities that are organized and managed together. There are five test levels:

- **Component testing (unit testing)**
  Testing components in isolation.
- **Component integration testing**
  Testing interactions between components.
- **System testing**
  Testing the overall behavior and capabilities of an entire system or product, including functional and non-functional testing.
- **System integration testing**
  Testing interfaces of the system under test and other systems and external services.
- **Acceptance testing**
  Focuses on validation and on demonstrating readiness for deployment, which means that the system fulfills the user's business needs.

Test Types
- **Functional testing** evaluates the functions that a component or system should perform. The functions are "what" the test object should do.
- **Non-functional testing** is the testing of "how well the system behaves". These include aspects such as performance efficiency, compatibility, and usability.
- **Black-box testing** is specification-based.
- **White-box testing** is structure-based.

Confirmation Testing and Regression Testing
- **Confirmation Testing** confirms that an original defect has been fixed.
- **Regression Testing** confirms that no adverse consequences have been caused by a change.

Maintenance Testing
The triggers for maintenance testing:
Modifications, Upgrades (Migrations), Retirement

3. Static Testing

Static Testing Basics
In contrast to dynamic testing, in static testing the software under test does not need to be executed. Static analysis can identify problems prior to dynamic testing while often requiring less effort, since no test cases are required, and tools are typically used.

Static Testing vs. Dynamic Testing
- Some defect types that can only be found by either static or dynamic testing.
- Static testing finds defects directly, while dynamic testing causes failures.
- Static testing may more easily detect defects that lay on paths through the code that are rarely executed.
- Static testing can be applied to non-executable work products.

Stakeholder Feedback
Early and frequent feedback allows for the early communication of potential quality problems.

Review Process Activities
- Planning
- Review Initiation
- Individual Review
- Communication and Analysis
- Fixing and Reporting

Roles and Responsibilities in Reviews
- **Manager** – decides what is to be reviewed and provides resources, such as staff and time
- **Author** – creates and fixes the work product
- **Moderator (Facilitator)** – ensures effective running of review meetings, including mediation, time management, and a safe environment
- **Scribe (Recorder)** – records anomalies and decisions during the review
- **Reviewer** – performs the review, may be a subject expert or stakeholder
- **Review leader** – takes responsibility such as deciding who will be involved, and organizing when and where the review will take place

Review Types
- Informal review – does not follow a defined process and does not require formal documented output.
- Walkthrough – led by the author, used for evaluating quality, building confidence, educating reviewers, and detecting anomalies.
- Technical Review – performed by technically qualified reviewers, led by a moderator. The goal is to make decisions regarding technical problems and detect anomalies.
- Inspection – the most formal type of review, following the complete generic process. The main objective is to find the maximum number of anomalies, evaluate quality, and improve the SDLC.

ISTQB - Certified Tester Foundation Level (CTFL) v4.0

4. Test Analysis and Design

Test Techniques

- **Black-box Test Techniques (specification-based)**
  Analysis of the specified behavior of the test object without reference to its internal structure. Therefore, the test cases are independent of how the software is implemented.
  - Equivalence Partitioning
  - Boundary Value Analysis
  - Decision Table Testing
  - State Transition Testing
- **White-box Test Techniques (structure-based)**
  Analysis of the test object's internal structure and processing. As the test cases are dependent on how the software is designed, they can only be created after the design or implementation of the test object.
  - Statement testing
  - Branch testing
- **Experience-based Test Techniques**
  Use the knowledge and experience of testers for the design and implementation of test cases. These test techniques depends on the tester's skills.
  - Error guessing
  - Exploratory testing
  - Checklist-based testing

Collaborative User Story Writing

A user story represents a feature that will be valuable to either a user or purchaser of a system or software. User stories have three critical aspects, called the "3 C's":

- **Card** – the medium describing a user story (e.g., an index card, an entry in an electronic board)
- **Conversation** – explains how the software will be used (can be documented or verbal)
- **Confirmation** – the acceptance criteria

The most common format for a user story is "As a [role], I want [goal to be accomplished], so that I can [resulting business value for the role]", followed by the acceptance criteria.

Acceptance Criteria

Acceptance criteria for a user story are the conditions that an implementation of the user story must meet to be accepted by stakeholders.

Acceptance Test-driven Development

ATDD is a test-first approach. Test cases are created prior to implementing the user story. The test cases are based on the acceptance criteria and can be seen as examples of how the software works. Examples and tests are the same.

5. Managing the Test Activities (1)

Test Plan

A test plan describes the test objectives, resources and processes for a test project.

Entry Criteria and Exit Criteria

Entry criteria define the preconditions for undertaking a given activity. Exit criteria define what must be achieved to declare an activity completed. Entry criteria and exit criteria should be defined for each test level, and will differ based on the test objectives.

Estimation Techniques

- **Estimation Based on Ratios**
- **Extrapolation**
- **Wideband Delphi** (e.g. Planning Poker)
  - Commonly used in Agile SW development
- **Three-Point Estimation**

Estimate:

$$E = \frac{a + 4m + b}{6} \quad \text{where:} \quad \begin{aligned} a &= \text{optimistic estimate} \\ m &= \text{most likely estimate} \\ b &= \text{pessimistic estimate} \end{aligned}$$

Measurement error:

$$SD = \frac{b - a}{6}$$

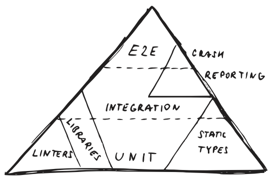
Test Case Prioritization

When prioritizing test cases, different factors can be taken into account. The most commonly used test case prioritization strategies are as follows:

- **Risk-based prioritization**
  Test cases covering the most important risks are executed first.
- **Coverage-based prioritization**
  Test cases achieving the highest coverage are executed first.
- **Requirements-based prioritization**
  Test cases related to the most important requirements are executed first.

Test Pyramid

The test pyramid model shows that different tests have different levels of granularity. Tests in the bottom layer are small, isolated, fast, and check small pieces of functionality, a lot of them are needed for coverage. The top layer represents complex, high-level, end-to-end tests.

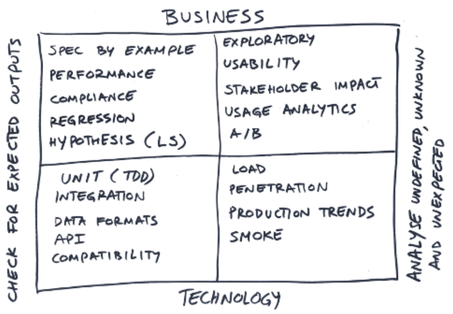


- UI tests, end-to-end (E2E) tests
- Service tests, Integration test
- Unit tests, Component tests

5. Managing the Test Activities (2)

Testing Quadrants

The testing quadrants group the test levels with the appropriate test types, activities, test techniques and work products in Agile software development.



- **Quadrant Q1** (technology facing, support the team): This quadrant contains component tests and component integration tests. These tests should be automated and included in the CI process.
- **Quadrant Q2** (business facing, support the team): This quadrant contains functional tests, examples, user story tests, user experience prototypes, API testing, and simulations. These tests check the acceptance criteria and can be manual or automated.
- **Quadrant Q3** (business facing, critique the product): This quadrant contains exploratory testing, usability testing, user acceptance testing. These tests are user-oriented and often manual.
- **Quadrant Q4** (technology facing, critique the product): This quadrant contains smoke tests and non-functional tests (except usability tests). These tests are often automated.

Risk Management

The main risk management activities are:

- **Risk analysis**: Risk identification and -assessment
- **Risk control** Risk mitigation and -monitoring

A risk can be characterized by two factors:

- Risk likelihood – probability of risk occurrence (0-1)
- Risk impact (harm) – consequences of occurrence

In software testing there are two types of risks:

- project risks - management issues
- product risks - product quality characteristics

Product Risk Analysis

Product risk analysis consists of **risk identification** and **risk assessment**. Risk assessment can use a quantitative or qualitative approach, or a mix of them.

- quantitative: risk level $=$ likelihood $*$ impact
- qualitative: risk level determined with risk matrix

5. Managing the Test Activities (3)

Monitoring, Control and Completion

- **Test Monitoring**
  Gathering information about testing.
- **Test Control**
  Uses the information from test monitoring to provide, in a form of the control directives, guidance and corrective actions to achieve better testing.
- **Test Completion**
  Collects data from completed test activities to consolidate experience, testware, and any other relevant information.

Metrics used in Testing

- Project progress metrics (e.g., task completion, resource usage, test effort).
- Test progress metrics (e.g., test case implementation progress, test environment preparation progress, number of test cases run/not run, passed/failed, test execution time).
- Product quality metrics (e.g., availability, response time, mean time to failure).
- Defect metrics (e.g., number and priorities of defects found/fixed, defect density, defect detection percentage).
- Risk metrics (e.g., residual risk level).
- Coverage metrics (e.g., requirements coverage, code coverage).
- Cost metrics (e.g., cost of testing, organizational cost of quality).

Configuration Management (CM)

CM provides a discipline for identifying, controlling, and tracking work products such as test plans, test strategies, test conditions, test cases, test scripts, test results, test logs, and test reports as configuration items.

Defect Management

Workflow for handling individual defects or anomalies from their discovery to their closure and rules for their classification.

Which of the test activities are MOST likely to involve the application of BVA and EP?

To identify the features that require testing, the test basis is analyzed and defined as test conditions, which are then prioritized along with related risks. The systematic identification of test conditions as coverage items often involves using **test techniques** both during test analysis and as part of the test design activity.

6. Test Tools

Tool Support for Testig

- **Test management tools**
  Tools to increase the test process efficiency by facilitating management of the SDLC, requirements, tests, defects, configuration
- **Static testing tools**
  Support tester in performing reviews and static analysis
- **Test design and test implementation tools**
  To generate test cases, data, and procedures
- **Test execution and test coverage tools**
  For automated test execution and coverage measurement
- **Non-functional testing tools**
  To perform non-functional testing that is difficult or impossible to perform manually
- **DevOps tools**
  Tools to support the DevOps delivery pipeline, workflow tracking, automated build processes, CI/CD
- **Collaboration tools** – facilitate communication
- **Tools supporting scalability and deployment standardization** (e.g., VMs, containerization tools)

Test Automation

Benefits of test automation:

- Provides **coverage measures** that are too complicated for humans to derive
- Time saved by reducing repetitive manual work
- Prevention of simple human errors
- More objective assessment
- Access to information (e.g., statistics, failure rates)
- Reduced test execution time

Risks of test automation:

- Unrealistic expectations
- Inaccurate estimations of time, costs, effort
- Using a test tool when manual testing is more appropriate
- Relying on a tool too much
- Dependency on tool vendor
- Problems with open-source software
- Incompatiblility of tool with the development platform
- Unsuitable tool (Compliance and standards)

Test activities / Testware

- **Test planning:** Consists of defining the test objectives and then selecting an approach that best achieves the objectives within the constraints.
- **Test monitoring and control:** Test monitoring involves the ongoing checking of all test activities and the comparison of actual progress against the plan. Test control involves taking the actions necessary to meet the objectives of testing.
- **Test analysis:** Includes analyzing the test basis to identify testable features and to define and prioritize associated test conditions (risk). The test basis and test objects are also evaluated to identify defects they may contain and to assess their testability. It is often supported by the use of test techniques and answers the question "what to test?".
  - **Testware:** prioritized test conditions (e.g., acceptance criteria), and defect reports for defects identified in the test basis
- **Test design:** Includes elaborating the test conditions into test cases and other testware (e.g., test charters). This activity often involves the identification of coverage items, which serve as a guide to specify test case inputs. Test techniques can be used to support this activity. Test design also includes defining the test data requirements, designing the test environment, and identifying any other required infrastructure and tools. It answers the question "how to test?".
  - **Testware:** prioritized test cases, test charters, coverage items, test data requirements, and test environment requirements
- **Test implementation:** Includes creating or acquiring the testware necessary for test execution (e.g., test data). Test cases can be organized into test procedures and are often assembled into test suites. Manual and automated test scripts are created. Test procedures are prioritized and arranged within a test execution schedule for efficient test execution. The test environment is built and verified.
  - **Testware:** test procedures, automated test scripts, test suites, test data, test execution schedule, and test environment elements such as stubs, drivers, simulators, and service virtualizations
- **Test execution:** Includes running the tests in accordance with the test execution schedule (test runs). Test execution may be manual or automated. Test execution can take many forms (continuous testing or pair testing sessions). Actual test results are compared with the expected results. The test results are logged. Anomalies are analyzed to identify their likely causes. This analysis allows us to report the anomalies based on the failures observed.
- **Test completion activities:** Occur at project milestones (e.g., release, end of iteration, test level completion)
  - **Testware:** test completion report, documented lessons learned, action items for improvement, and change requests (as product backlog items)

4.2. Black-Box Test Techniques

Basics

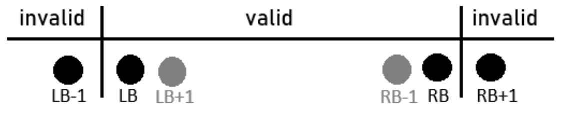
Black-box test technique, are based on an analysis of the specified behavior of the test object without reference to its internal structure. In black-box testing (specification-based) the test cases are based on an analysis of business logic, test cases are **not** based on the decisions in the source code. Therefore, test cases are independent of how the software is implemented.

1. Eqivalence Partitioning (EP)

EP divides data into partitions. If a test case, that tests one value from an equivalence partition, detects a defect, this defect should also be detected by test cases that test any other value from the same partition. Therefore, one test for each partition is sufficient.

2. Boundary Value Analysis (BVA)

BVA is a test technique based on exercising the boundaries of equivalence partitions. Therefore, BVA can only be used for ordered partitions. The minimum value (left boundary (LB))and maximum value (right boundary (RB)) of a partition are its boundary values. In the case of BVA, if two elements belong to the same partition, all elements between them must also belong to that partition.



- 2-value BVA: LB-1, LB, RB, RB+1
- 3-value BVA: LB-1, LB, LB+1, RB-1, RB, RB+1

3. Decision Table Testing

Decision tables are used for testing the implementation of requirements that specify how different combinations of conditions result in different outcomes. Decision tables are an effective way of recording complex logic, such as business rules.

4. State Transition Testing

A state diagram models the behavior of a system by showing its possible states and valid state transitions. A transition is initiated by an event, which may be additionally qualified by a guard condition. The transitions are assumed to be instantaneous and may sometimes result in the software taking action. The common transition labeling syntax is as follows: "event [guard condition] / action". Guard conditions and actions can be omitted if they do not exist or are irrelevant for the tester.

- all states coverage
- valid transition coverage
- all transitions coverage

All states coverage is weaker than valid transitions coverage, because it can typically be achieved without exercising all the transitions.

4.3. White-Box Test Techniques

Basics

In white-box test techniques (structure-based testing), test cases are derived based on the source code structure. If a test case is based on the knowledge of the control flow of the test object, it is a white-box test technique. Test cases are based on an analysis of the test object's internal structure and processing. As the test cases are dependent on how the software is designed and coded, they can only be created after the design or implementation of the test object.

Value of White-box Testing

White-box test techniques can be used in static testing (e.g., during dry runs of code). They are well suited to reviewing code not yet ready for execution, pseudocode and other high-level or top-down logic which can be modeled with a control flow graph.

1. Statement Testing and Statement Coverage

In statement testing, the coverage items are executable statements. The aim is to design test cases that exercise statements in the code until an acceptable level of coverage is achieved. Coverage is measured as the number of statements exercised by the test cases divided by the total number of executable statements in the code, and is expressed as a percentage.

2. Branch Testing and Branch Coverage

In branch testing, the test cases are derived from knowledge of the control flow of the test object.
A branch is a transfer of control between two nodes in the control flow graph, which shows the possible sequences in which source code statements are executed in the test object. Each transfer of control can be either **unconditional** (i.e., straight-line code) or **conditional** (i.e., a decision outcome).

Branch coverage metric

In branch testing the coverage items are branches, which are represented by the edges of a control flow graph. The formula for branch coverage B_{Cov} is given by:

$$B_{\text{Cov}} = \left(\frac{\text{branches exercised by test cases}}{\text{total number of branches in code}}\right) \times 100\%$$

Branch coverage vs statement coverage

Branch coverage subsumes statement coverage. This means that any set of test cases achieving 100% branch coverage also achieves 100% statement coverage (but not vice versa).

Statement coverage

Test T1 achieved 40% statement coverage and test T2 achieved 65% statement coverage.
The statements executed by T1 and T2 were disjoint, the coverage of the test suite T1, T2 would be 105%, which is impossible. At least 5% of executable statements must have been executed by both T1 and T2.

4.4. Experience-based Test Techniques

1. Error guessing

Error guessing is a test technique used to anticipate the occurrence of errors, defects, and failures, based on the tester's knowledge, including:

- How the application has worked in the past
- Types of errors the developers tend to make
- Types of failures in similar applications

Fault attack is a methodical approach to the implementation of error guessing and requires a checklist of possible errors, defects and failures, and to design tests that will identify defects associated with the errors, expose the defects, or cause the failures. Both, error guessing and checklist-based testing use lists.

2. Exploratory testing

In exploratory testing, tests are simultaneously designed, executed, and evaluated while the tester learns about the test object. The testing is used to learn more about the test object, to explore it more deeply with focused tests, and to create tests for untested areas.

Exploratory testing is most useful when there are few known specifications and/or there is a pressing timeline for testing (e.g., delay in the release of a brand-new application).

3. Checklist-based testing

In checklist-based testing, a tester designs, implements, and executes tests to cover test conditions from a checklist. Checklists can be built based on experience, knowledge about what is important for the user, or an understanding of why and how software fails. Checklists should not contain items that can be checked automatically, items better suited as entry criteria, exit criteria, or items that are too general.

Acceptance Criteria

Format

Set of predefined requirements or conditions that a product, feature, or functionality must meet to be considered complete and acceptable to stakeholders.

- Scenario-Oriented Format (Given-When-Then)
  Describes a scenario to be verified.
- Rules-Based Format
  Includes formats like bullet point verification lists or tabulated forms of input-output mappings, explicitly showing the rules to be followed.

Validation vs Verification

- **Validation** is concerned with meeting user requirements and expectations
- **Verification** is concerned with meeting specified requirements

Acceptance testing focuses on validation and on demonstrating readiness for deployment, which means that the system fulfills the user's business needs. Should be performed by the intended users.

Reviews

Review activities

- **Planning**
  Defining the review scope, purpose, work product to be reviewed, quality characteristics to be evaluated, areas of focus, exit criteria, supporting information such as standards, effort, and timeframes.
- **Review initiation**
  Ensuring all participants have access to the work product and necessary resources, and clarifying their roles and responsibilities.
- **Individual review**
  Evaluating the work product's quality, identifying and logging anomalies, recommendations, and questions using review techniques like checklist-based and scenario-based reviewing.
- **Communication and analysis**
  Analyzing and discussing each anomaly, determining its status, ownership, and required actions, and making review decisions, normally in a meeting. This could include determining the need for a follow-up review.

Roles in reviews

- **Scribe (Recorder)**
  Responsible for gathering feedback and recording information from reviewers and documenting review information, such as decisions made, and any new anomalies identified during the review meeting.
- **Review Leader**
  Responsible for overseeing the review process, such as selecting the review team members, scheduling review meetings (organizing when and where), and ensuring that the review is completed successfully.
- **Facilitator (Moderator)**
  Responsible for ensuring that the review meetings run effectively, including managing time, mediating discussions, and creating a safe environment where everyone can voice their opinions freely.
- **Manager**
  Responsible for deciding what needs to be reviewed and allocating resources, such as staff and time.

Anomalies

Def. "Anomalies"

Deviations from expected behavior in software:

- (real) defects or bugs - errors in the code
- failures - system does not perform as intended
- errors - mistakes made by developers, ...
- false-positives – non-issues flagged as problems
- change request

Keywords (K1)

Overview

All keywords shall be remembered (K1).

1. Fundamentals of Testing

coverage, debugging, defect, error, failure, quality, quality assurance, root cause, test analysis, test basis, test case, test completion, test condition, test control, test data, test design, test execution, test implementation, test monitoring, test object, test objective, test planning, test procedure, test process, test result, testing, testware, traceability, validation, verification

2. Testing Throughout the SDLC

acceptance testing, black-box testing, component integration testing, component testing, confirmation testing, functional testing, integration testing, maintenance testing, non-functional testing, regression testing, shift left, system integration testing, system testing, test level, test object, test type, white-box testing

3. Static Testing

anomaly, dynamic testing, formal review, informal review, inspection, review, static analysis, static testing, technical review, walkthrough

4. Test Analysis and Design

acceptance criteria, acceptance test-driven development, black-box test technique, boundary value analysis, branch coverage, checklist-based testing, collaboration-based test approach, coverage, coverage item, decision table testing, equivalence partitioning, error guessing, experience-based test technique, exploratory testing, state transition testing, statement coverage, test technique, white-box test technique

5. Managing the Test Activities

defect management, defect report, entry criteria, exit criteria, product risk, project risk, risk, risk analysis, risk assessment, risk control, risk identification, risk level, risk management, risk mitigation, risk monitoring, risk-based testing, test approach, test completion report, test control, test monitoring, test plan, test planning, test progress report, test pyramid, test strategy, testing quadrants

6. Test Tools

test automation

Certification

ISTQB

The International Software Testing Qualifications Board (ISTQB) is a software testing certification board.

Overview

The ISTQB Certified Tester Foundation Level (CTFL) certification is the cornerstone of essential testing knowledge that can be applied to real-world scenarios.

Exam Structure

- **No. of Questions**: 40
- **Passing Score**: 26
- **Total Points**: 40
- **Time (min)**: 60 (+15 min for Non-Native Language)

Learning objectives levels

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

Exam Structures Rules

K-Level	No. of Questions	Time	Total Time
K1	8	1	8
K2	24	1	24
K3	8	3	24
Total	**40**		**56 min**

K-Level Breakdown and Total Time

Chapter	K1	K2	K3	Points	Percentage
Chapter 1	2	6	0	8	**13%**
Chapter 2	1	4	0	5	**8%**
Chapter 3	1	3	1	5	**8%**
Chapter 4	1	5	5	11	**18%**
Chapter 5	2	5	2	9	**15%**
Chapter 6	1	1	0	2	**3%**
Total	10	24	8	60	100%

Question Breakdown by Chapter and K-Level