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 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

The principle of early testing is sometimes referred to as shift left because it is an approach where testing is performed earlier in the SDLC.

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.

Maintenance Testing

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

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.

Early and Frequent 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 to review and provides resources
- Author – creates and fixes the work product
- Moderator (also known as the facilitator) – runs review meetings and ensures a safe environment
- Scribe (also known as recorder) – records anomalies and decisions during the review
- Reviewer – performs the review, may be a subject expert or stakeholder
- Review leader – takes overall responsibility for review

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.

Antonia Frey · 23-12-2024

4. Test Analysis and Design

Test Techniques

- **Black-box Test Techniques**
  Black-box test techniques (also known as specification-based techniques) are based on an 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**
  White-box test techniques (also known as structure-based techniques) 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, they can only be created after the design or implementation of the test object.
  - Statement testing
  - Branch testing
- **Experience-based Test Techniques**
  Experience-based test techniques effectively 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)

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

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:

$$E = \frac{a + 4m + b}{6}$$ where: a = optimistic estimate, m = most likely estimate, b = pessimistic estimate

Measurement error:

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

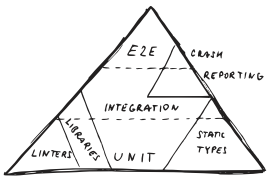
Test Case Prioritization

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

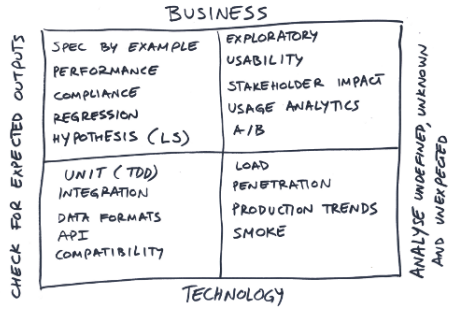
Tests in the bottom layer are small, isolated, fast, and check small pieces of functionality, a lot of them are needed for coverage. The top layer represents complex, high-level, end-to-end tests.



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

Testing Quadrants

The testing quadrants, defined by Brian Marick (Marick 2003, Crispin 2008), group the test levels with the appropriate test types, activities, test techniques and work products in the 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

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.

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**
  Tools to support the tester in performing reviews and static analysis
- **Test design and test implementation tools**
  Tools to generate test cases, test data, and test procedures
- **Test execution and test coverage tools**
  Tools for automated test execution and coverage measurement
- **Non-functional testing tools**
  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)
- **Other tools**
  (e.g., a spreadsheet is a tool in the context of testing)

Benefits and Risks of Test Automation

Benefits of test automation:

- 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
- Choosing an unsuitable tool (Compliance and standards)

Certification

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

Keywords (K1)