1. Fundamentals of Testing – 180 minutes

Keywords

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 result, testing, testware, validation, verification

Learning Objectives for Chapter 1:

1.1 What is Testing?

FL-1.1.1 (K1) Identify typical test objectives

FL-1.1.2 (K2) Differentiate testing from debugging

1.2 Why is Testing Necessary?

FL-1.2.1 (K2) Exemplify why testing is necessary

FL-1.2.2 (K1) Recall the relation between testing and quality assurance

FL-1.2.3 (K2) Distinguish between root cause, error, defect, and failure

1.3 Testing Principles

FL-1.3.1 (K2) Explain the seven testing principles

1.4 Test Activities, Testware and Test Roles

FL-1.4.1 (K2) Summarize the different test activities and tasks

FL-1.4.2 (K2) Explain the impact of context on the test process

FL-1.4.3 (K2) Differentiate the testware that supports the test activities

FL-1.4.4 (K2) Explain the value of maintaining traceability

FL-1.4.5 (K2) Compare the different roles in testing

1.5 Essential Skills and Good Practices in Testing

FL-1.5.1 (K2) Give examples of the generic skills required for testing

FL-1.5.2 (K1) Recall the advantages of the whole team approach

FL-1.5.3 (K2) Distinguish the benefits and drawbacks of independence of testing

1.1. What is Testing?

Software systems are an integral part of our daily life. Most people have had 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, in extreme cases, even injury or death. Software testing assesses software quality and helps reducing the risk of software failure in operation.

Software testing is a set of activities to discover defects and evaluate the quality of software artifacts. These artifacts, when being tested, are known as test objects. A common misconception about testing is that it only consists of executing tests (i.e., running the software and checking the test results). However, software testing also includes other activities and must be aligned with the software development lifecycle (see chapter 2).

Another common misconception about testing is that testing focuses entirely on verifying the test object. Whilst testing involves verification, i.e., checking whether the system meets specified requirements, it also involves validation, which means checking whether the system meets users' and other stakeholders' needs in its operational environment.

Testing may be dynamic or static. Dynamic testing involves the execution of software, while static testing does not. Static testing includes reviews (see chapter 3) and static analysis. Dynamic testing uses different types of test techniques and test approaches to derive test cases (see chapter 4).

Testing is not only a technical activity. It also needs to be properly planned, managed, estimated, monitored and controlled (see chapter 5).

Testers use tools (see chapter 6), but it is important to remember that testing is largely an intellectual activity, requiring the testers to have specialized knowledge, use analytical skills and apply critical thinking and systems thinking (Myers 2011, Roman 2018).

The ISO/IEC/IEEE 29119-1 standard provides further information about software testing concepts.

1.1.1. Test Objectives

The typical test objectives are:

- Evaluating work products such as requirements, user stories, designs, and code
- Triggering failures and finding defects
- Ensuring required coverage of a test object
- Reducing the level of risk of inadequate software quality
- Verifying whether specified requirements have been fulfilled
- Verifying that a test object complies with contractual, legal, and regulatory requirements
- Providing information to stakeholders to allow them to make informed decisions
- Building confidence in the quality of the test object
- Validating whether the test object is complete and works as expected by the stakeholders

Objectives of testing can vary, depending upon the context, which includes the work product being tested,

the test level, risks, the software development lifecycle (SDLC) being followed, and factors related to the business context, e.g., corporate structure, competitive considerations, or time to market.

1.1.2. Testing and Debugging

Testing and debugging are separate activities. 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).

When dynamic testing (see chapter 4) triggers a failure, debugging is concerned with finding causes of this failure (defects), analyzing these causes, and eliminating them. The typical debugging process in this case involves:

- Reproduction of a failure
- Diagnosis (finding the root cause)
- Fixing the cause

Subsequent confirmation testing checks whether the fixes resolved the problem. Preferably, confirmation testing is done by the same person who performed the initial test. Subsequent regression testing can also be performed, to check whether the fixes are causing failures in other parts of the test object (see section 2.2.3 for more information on confirmation testing and regression testing).

When static testing identifies a defect, debugging is concerned with removing it. There is no need for reproduction or diagnosis, since static testing directly finds defects, and cannot cause failures (see chapter 3).

1.2. Why is Testing Necessary?

Testing, as a form of quality control, helps in achieving the agreed upon goals within the set scope, time, quality, and budget constraints. Testing's contribution to success should not be restricted to the test team activities. Any stakeholder can use their testing skills to bring the project closer to success. Testing components, systems, and associated documentation helps to identify defects in software,

1.2.1. Testing's Contributions to Success

Testing provides a cost-effective means of detecting defects. These defects can then be removed (by debugging – a non-testing activity), so testing indirectly contributes to higher quality test objects.

Testing provides a means of directly evaluating the quality of a test object at various stages in the SDLC. These measures are used as part of a larger project management activity, contributing to decisions to move to the next stage of the SDLC, such as the release decision.

Testing provides users with indirect representation on the development project. Testers ensure that their understanding of users' needs are considered throughout the development lifecycle. The alternative is to involve a representative set of users as part of the development project, which is not usually possible due to the high costs and lack of availability of suitable users.

Testing may also be required to meet contractual or legal requirements, or to comply with regulatory standards.

1.2.2. Testing and Quality Assurance (QA)

While people often use the terms “testing” and “quality assurance” (QA) interchangeably, testing and QA are not the same. Testing is a form of quality control (QC)
QC is a product-oriented, corrective approach that focuses on those activities supporting the achievement of appropriate levels of quality. Testing is a major form of quality control, while others include formal methods (model checking and proof of correctness), simulation and prototyping.

QA is a process-oriented, preventive approach that focuses on the implementation and improvement of processes. It works on the basis that if a good process is followed correctly, then it will generate a good product. QA applies to both the development and testing processes, and is the responsibility of everyone on a project.

Test results are used by QA and QC. In QC they are used to fix defects, while in QA they provide feedback on how well the development and test processes are performing.

1.2.3. Errors, Defects, Failures, and Root Causes

Human beings make errors (mistakes), which produce defects (faults, bugs), which in turn may result in failures. Humans make errors for various reasons, such as time pressure, complexity of work products, processes, infrastructure or interactions, or simply because they are tired or lack adequate training.

Defects can be found in documentation, such as a requirements specification or a test script, in source code, or in a supporting artifact such as a build file. Defects in artifacts produced earlier in the SDLC, if undetected, often lead to defective artifacts later in the lifecycle. If a defect in code is executed, the system may fail to do what it should do, or do something it shouldn’t, causing a failure. Some defects will always result in a failure if executed, while others will only result in a failure in specific circumstances, and some may never result in a failure.

Errors and defects are not the only cause of failures. Failures can also be caused by environmental conditions, such as when radiation or electromagnetic field cause defects in firmware.

A root cause is a fundamental reason for the occurrence of a problem (e.g., a situation that leads to an error). Root causes are identified through root cause analysis, which is typically performed when a failure occurs or a defect is identified. It is believed that further similar failures or defects can be prevented or their frequency reduced by addressing the root cause, such as by removing it.

1.3. Testing Principles

A number of testing principles offering general guidelines applicable to all testing have been suggested over the years. This syllabus describes seven such principles.

1. Testing shows the presence, not the absence of defects. Testing can show that defects are present in the test object, but cannot prove that there are no defects (Buxton 1970). Testing reduces the probability of defects remaining undiscovered in the test object, but even if no defects are found, testing cannot prove test object correctness.

2. Exhaustive testing is impossible. Testing everything is not feasible except in trivial cases (Manna 1978). Rather than attempting to test exhaustively, test techniques (see chapter 4), test case prioritization (see section 5.1.5), and risk-based testing (see section 5.2), should be used to focus test efforts.

3. Early testing saves time and money. Defects that are removed early in the process will not cause subsequent defects in derived work products. The cost of quality will be reduced since fewer failures will

occur later in the SDLC (Boehm 1981). To find defects early, both static testing (see chapter 3) and dynamic testing (see chapter 4) should be started as early as possible.

4. Defects cluster together. A small number of system components usually contain most of the defects discovered or are responsible for most of the operational failures (Enders 1975). This phenomenon is an illustration of the Pareto principle. Predicted defect clusters, and actual defect clusters observed during testing or in operation, are an important input for risk-based testing (see section 5.2).

5. Tests wear out. If the same tests are repeated many times, they become increasingly ineffective in detecting new defects (Beizer 1990). To overcome this effect, existing tests and test data may need to be modified, and new tests may need to be written. However, in some cases, repeating the same tests can have a beneficial outcome, e.g., in automated regression testing (see section 2.2.3).

6. Testing is context dependent. There is no single universally applicable approach to testing. Testing is done differently in different contexts (Kaner 2011).

7. Absence-of-defects fallacy. It is a fallacy (i.e., a misconception) to expect that software verification will ensure the success of a system. Thoroughly testing all the specified requirements and fixing all the defects found could still produce a system that does not fulfill the users' needs and expectations, that does not help in achieving the customer's business goals, and that is inferior compared to other competing systems. In addition to verification, validation should also be carried out (Boehm 1981).

1.4. Test Activities, Testware and Test Roles

Testing is context dependent, but, at a high level, there are common sets of test activities without which testing is less likely to achieve test objectives. These sets of test activities form a test process. The test process can be tailored to a given situation based on various factors. Which test activities are included in this test process, how they are implemented, and when they occur is normally decided as part of the test planning for the specific situation (see section 5.1).

The following sections describe the general aspects of this test process in terms of test activities and tasks, the impact of context, testware, traceability between the test basis and testware, and testing roles.

The ISO/IEC/IEEE 29119-2 standard provides further information about test processes.

1.4.1. Test Activities and Tasks

A test process usually consists of the main groups of activities described below. Although many of these activities may appear to follow a logical sequence, they are often implemented iteratively or in parallel. These testing activities usually need to be tailored to the system and the project.

Test planning consists of defining the test objectives and then selecting an approach that best achieves the objectives within the constraints imposed by the overall context. Test planning is further explained in section 5.1.

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 monitoring and control are further explained in section 5.3.

Test analysis includes analyzing the test basis to identify testable features and to define and prioritize associated test conditions, together with the related risks and risk levels (see section 5.2). The test basis and the test objects are also evaluated to identify defects they may contain and to assess their testability.

Test analysis is often supported by the use of test techniques (see chapter 4). Test analysis answers the question “what to test?” in terms of measurable coverage criteria.

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 (see chapter 4) 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. Test design answers the question “how to test?”.

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 (see section 5.1.5). The test environment is built and verified to be set up correctly.

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, including 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 (see section 5.5).

Test completion activities usually occur at project milestones (e.g., release, end of iteration, test level completion) for any unresolved defects, change requests or product backlog items created. Any testware that may be useful in the future is identified and archived or handed over to the appropriate teams. The test environment is shut down to an agreed state. The test activities are analyzed to identify lessons learned and improvements for future iterations, releases, or projects (see section 2.1.6). A test completion report is created and communicated to the stakeholders.

1.4.2. Test Process in Context

Testing is not performed in isolation. Test activities are an integral part of the development processes carried out within an organization. Testing is also funded by stakeholders and its final goal is to help fulfill the stakeholders’ business needs. Therefore, the way the testing is carried out will depend on a number of contextual factors including:

- Stakeholders (needs, expectations, requirements, willingness to cooperate, etc.)
- Team members (skills, knowledge, level of experience, availability, training needs, etc.)
- Business domain (criticality of the test object, identified risks, market needs, specific legal regulations, etc.)
- Technical factors (type of software, product architecture, technology used, etc.)
- Project constraints (scope, time, budget, resources, etc.)
- Organizational factors (organizational structure, existing policies, practices used, etc.)
- Software development lifecycle (engineering practices, development methods, etc.)
- Tools (availability, usability, compliance, etc.)

These factors will have an impact on many test-related issues, including: test strategy, test techniques

used, degree of test automation, required level of coverage, level of detail of test documentation, reporting, etc.

1.4.3. Testware

Testware is created as output work products from the test activities described in section 1.4.1. There is a significant variation in how different organizations produce, shape, name, organize and manage their work products. Proper configuration management (see section 5.4) ensures consistency and integrity of work products. The following list of work products is not exhaustive:

- **Test planning work products** include: test plan, test schedule, risk register, and entry and exit criteria (see section 5.1). Risk register is a list of risks together with risk likelihood, risk impact and information about risk mitigation (see section 5.2). Test schedule, risk register and entry and exit criteria are often a part of the test plan.
- **Test monitoring and control work products** include: test progress reports (see section 5.3.2), documentation of control directives (see section 5.3) and risk information (see section 5.2).
- **Test analysis work products** include: (prioritized) test conditions (e.g., acceptance criteria, see section 4.5.2), and defect reports regarding defects in the test basis (if not fixed directly).
- **Test design work products** include: (prioritized) test cases, test charters, coverage items, test data requirements and test environment requirements.
- **Test implementation work products** include: test procedures, automated test scripts, test suites, test data, test execution schedule, and test environment elements. Examples of test environment elements include: stubs, drivers, simulators, and service virtualizations.
- **Test execution work products** include: test logs, and defect reports (see section 5.5).
- **Test completion work products** include: test completion report (see section 5.3.2), action items for improvement of subsequent projects or iterations, documented lessons learned, and change requests (e.g., as product backlog items).

1.4.4. Traceability between the Test Basis and Testware

In order to implement effective test monitoring and control, it is important to establish and maintain traceability throughout the test process between the test basis elements, testware associated with these elements (e.g., test conditions, risks, test cases), test results, and detected defects.

Accurate traceability supports coverage evaluation, so it is very useful if measurable coverage criteria are defined in the test basis. The coverage criteria can function as key performance indicators to drive the activities that show to what extent the test objectives have been achieved (see section 1.1.1). For example:

- Traceability of test cases to requirements can verify that the requirements are covered by test cases.
- Traceability of test results to risks can be used to evaluate the level of residual risk in a test object.

In addition to evaluating coverage, good traceability makes it possible to determine the impact of

changes, facilitates test audits, and helps meet IT governance criteria. Good traceability also makes test progress and completion reports more easily understandable by including the status of test basis elements. This can also assist in communicating the technical aspects of testing to stakeholders in an understandable manner. Traceability provides information to assess product quality, process capability, and project progress against business goals.

1.4.5. Roles in Testing

In this syllabus, two principal roles in testing are covered: a test management role and a testing role. The activities and tasks assigned to these two roles depend on factors such as the project and product context, the skills of the people in the roles, and the organization.

The test management role takes overall responsibility for the test process, test team and leadership of the test activities. The test management role is mainly focused on the activities of test planning, test monitoring and control and test completion. The way in which the test management role is carried out varies depending on the context. For example, in Agile software development, some of the test management tasks may be handled by the Agile team. Tasks that span multiple teams or the entire organization may be performed by test managers outside of the development team.

The testing role takes overall responsibility for the engineering (technical) aspect of testing. The testing role is mainly focused on the activities of test analysis, test design, test implementation and test execution.

Different people may take on these roles at different times. For example, the test management role can be performed by a team leader, by a test manager, by a development manager, etc. It is also possible for one person to take on the roles of testing and test management at the same time.

1.5. Essential Skills and Good Practices in Testing

Skill is the ability to do something well that comes from one's knowledge, practice and aptitude. Good testers should possess some essential skills to do their job well. Good testers should be effective team players and should be able to perform testing on different levels of test independence.

1.5.1. Generic Skills Required for Testing

While being generic, the following skills are particularly relevant for testers:

- Testing knowledge (to increase effectiveness of testing, e.g., by using test techniques)
- Thoroughness, carefulness, curiosity, attention to details, being methodical (to identify defects, especially the ones that are difficult to find)
- Good communication skills, active listening, being a team player (to interact effectively with all stakeholders, to convey information to others, to be understood, and to report and discuss defects)
- Analytical thinking, critical thinking, creativity (to increase effectiveness of testing)
- Technical knowledge (to increase efficiency of testing, e.g., by using appropriate test tools)
- Domain knowledge (to be able to understand and to communicate with end users/business representatives)

Testers are often the bearers of bad news. It is a common human trait to blame the bearer of bad news. This makes communication skills crucial for testers. Communicating test results may be perceived as criticism of the product and of its author. Confirmation bias can make it difficult to accept information that disagrees with currently held beliefs. Some people may perceive testing as a destructive activity, even though it contributes greatly to project success and product quality. To try to improve this view, information about defects and failures should be communicated in a constructive way.

1.5.2. Whole Team Approach

One of the important skills for a tester is the ability to work effectively in a team context and to contribute positively to the team goals. The whole team approach – a practice coming from Extreme Programming (see section 2.1) – builds upon this skill.

In the whole-team approach any team member with the necessary knowledge and skills can perform any task, and everyone is responsible for quality. The team members share the same workspace (physical or virtual), as co-location facilitates communication and interaction. The whole team approach improves team dynamics, enhances communication and collaboration within the team, and creates synergy by allowing the various skill sets within the team to be leveraged for the benefit of the project.

Testers work closely with other team members to ensure that the desired quality levels are achieved. This includes collaborating with business representatives to help them create suitable acceptance tests and working with developers to agree on the test strategy and decide on test automation approaches. Testers can thus transfer testing knowledge to other team members and influence the development of the product.

Depending on the context, the whole team approach may not always be appropriate. For instance, in some situations, such as safety-critical, a high level of test independence may be needed.

1.5.3. Independence of Testing

A certain degree of independence makes the tester more effective at finding defects due to differences between the author's and the tester's cognitive biases (cf. Salman 1995). Independence is not, however, a replacement for familiarity, e.g., developers can efficiently find many defects in their own code.

Work products can be tested by their author (no independence), by the author's peers from the same team (some independence), by testers from outside the author's team but within the organization (high independence), or by testers from outside the organization (very high independence). For most projects, it is usually best to carry out testing with multiple levels of independence (e.g., developers performing component and component integration testing, test team performing system and system integration testing, and business representatives performing acceptance testing).

The main benefit of independence of testing is that independent testers are likely to recognize different kinds of failures and defects compared to developers because of their different backgrounds, technical perspectives, and biases. Moreover, an independent tester can verify, challenge, or disprove assumptions made by stakeholders during specification and implementation of the system.

However, there are also some drawbacks. Independent testers may be isolated from the development team, which may lead to a lack of collaboration, communication problems, or an adversarial relationship with the development team. Developers may lose a sense of responsibility for quality. Independent testers may be seen as a bottleneck or be blamed for delays in release.