1.1 Why is Testing Necessary (K2) — *20 minutes*

Terms
Bug, defect, error, failure, fault, mistake, quality, risk

1.1.1 Software Systems Context (K1)

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 could even cause injury or death.

1.1.2 Causes of Software Defects (K2)

A human being can make an error (mistake), which produces a defect (fault, bug) in the program code, or in a document. 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. Defects in software, systems or documents may result in failures, but not all defects do so.

Defects occur because human beings are fallible and because there is time pressure, complex code, complexity of infrastructure, changing technologies, and/or many system interactions.

Failures can be caused by environmental conditions as well. For example, radiation, magnetism, electronic fields, and pollution can cause faults in firmware or influence the execution of software by changing the hardware conditions.

1.1.3 Role of Testing in Software Development, Maintenance and Operations (K2)

Rigorous testing of systems and documentation can help to reduce the risk of problems occurring during operation and contribute to the quality of the software system, if the defects found are corrected before the system is released for operational use.

Software testing may also be required to meet contractual or legal requirements, or industry-specific standards.

1.1.4 Testing and Quality (K2)

With the help of testing, it is possible to measure the quality of software in terms of defects found, for both functional and non-functional software requirements and characteristics (e.g., reliability, usability, efficiency, maintainability and portability). For more information on non-functional testing see Chapter 2; for more information on software characteristics see 'Software Engineering – Software Product Quality' (ISO 9126).

Testing can give confidence in the quality of the software if it finds few or no defects. A properly designed test that passes reduces the overall level of risk in a system. When testing does find defects, the quality of the software system increases when those defects are fixed.

Lessons should be learned from previous projects. By understanding the root causes of defects found in other projects, processes can be improved, which in turn should prevent those defects from reoccurring and, as a consequence, improve the quality of future systems. This is an aspect of quality assurance.

Testing should be integrated as one of the quality assurance activities (i.e., alongside development standards, training and defect analysis).

1.1.5 How Much Testing is Enough? (K2)

Deciding how much testing is enough should take account of the level of risk, including technical, safety, and business risks, and project constraints such as time and budget. Risk is discussed further in Chapter 5.

Testing should provide sufficient information to stakeholders to make informed decisions about the release of the software or system being tested, for the next development step or handover to customers.

1.2 What is Testing? (K2) — *30 minutes*

Terms
Debugging, requirement, review, test case, testing, test objective

Background
A common perception of testing is that it only consists of running tests, i.e., executing the software. This is part of testing, but not all of the testing activities.

Test activities exist before and after test execution. These activities include planning and control, choosing test conditions, designing and executing test cases, checking results, evaluating exit criteria, reporting on the testing process and system under test, and finalizing or completing closure activities after a test phase has been completed. Testing also includes reviewing documents (including source code) and conducting static analysis.

Both dynamic testing and static testing can be used as a means for achieving similar objectives, and will provide information that can be used to improve both the system being tested and the development and testing processes.

Testing can have the following objectives:
- Finding defects
- Gaining confidence about the level of quality
- Providing information for decision-making
- Preventing defects

The thought process and activities involved in designing tests early in the life cycle (verifying the test basis via test design) can help to prevent defects from being introduced into code. Reviews of documents (e.g., requirements) and the identification and resolution of issues also help to prevent defects appearing in the code.

Different viewpoints in testing take different objectives into account. For example, in development testing (e.g., component, integration and system testing), the main objective may be to cause as many failures as possible so that defects in the software are identified and can be fixed. In acceptance testing, the main objective may be to confirm that the system works as expected, to gain confidence that it has met the requirements. In some cases the main objective of testing may be to assess the quality of the software (with no intention of fixing defects), to give information to stakeholders of the risk of releasing the system at a given time. Maintenance testing often includes testing that no new defects have been introduced during development of the changes. During operational testing, the main objective may be to assess system characteristics such as reliability or availability.

Debugging and testing are different. Dynamic testing can show failures that are caused by defects. Debugging is the development activity that finds, analyzes and removes the cause of the failure. Subsequent re-testing by a tester ensures that the fix does indeed resolve the failure. The responsibility for these activities is usually testers test and developers debug.

The process of testing and the testing activities are explained in Section 1.4.

1.3 Seven Testing Principles (K2) — *35 minutes*

Terms
Exhaustive testing

Principles
A number of testing principles have been suggested over the past 40 years and offer general guidelines common for all testing.

Principle 1 – Testing shows presence of defects
Testing can show that defects are present, but cannot prove that there are no defects. Testing reduces the probability of undiscovered defects remaining in the software but, even if no defects are found, it is not a proof of correctness.

Principle 2 – Exhaustive testing is impossible
Testing everything (all combinations of inputs and preconditions) is not feasible except for trivial cases. Instead of exhaustive testing, risk analysis and priorities should be used to focus testing efforts.

Principle 3 – Early testing
To find defects early, testing activities shall be started as early as possible in the software or system development life cycle, and shall be focused on defined objectives.

Principle 4 – Defect clustering
Testing effort shall be focused proportionally to the expected and later observed defect density of modules. A small number of modules usually contains most of the defects discovered during pre-release testing, or is responsible for most of the operational failures.

Principle 5 – Pesticide paradox
If the same tests are repeated over and over again, eventually the same set of test cases will no longer find any new defects. To overcome this “pesticide paradox”, test cases need to be regularly reviewed and revised, and new and different tests need to be written to exercise different parts of the software or system to find potentially more defects.

Principle 6 – Testing is context dependent
Testing is done differently in different contexts. For example, safety-critical software is tested differently from an e-commerce site.

Principle 7 – Absence-of-errors fallacy
Finding and fixing defects does not help if the system built is unusable and does not fulfill the users’ needs and expectations.

1.4 Fundamental Test Process (K1) — *35 minutes*

Terms
Confirmation testing, re-testing, exit criteria, incident, regression testing, test basis, test condition, test coverage, test data, test execution, test log, test plan, test procedure, test policy, test suite, test summary report, testware

Background
The most visible part of testing is test execution. But to be effective and efficient, test plans should also include time to be spent on planning the tests, designing test cases, preparing for execution and evaluating results.

The fundamental test process consists of the following main activities:
- Test planning and control
- Test analysis and design
- Test implementation and execution
- Evaluating exit criteria and reporting
- Test closure activities

Although logically sequential, the activities in the process may overlap or take place concurrently. Tailoring these main activities within the context of the system and the project is usually required.

1.4.1 Test Planning and Control (K1)
Test planning is the activity of defining the objectives of testing and the specification of test activities in order to meet the objectives and mission.

Test control is the ongoing activity of comparing actual progress against the plan, and reporting the status, including deviations from the plan. It involves taking actions necessary to meet the mission and objectives of the project. In order to control testing, the testing activities should be monitored throughout the project. Test planning takes into account the feedback from monitoring and control activities.

Test planning and control tasks are defined in Chapter 5 of this syllabus.

1.4.2 Test Analysis and Design (K1)
Test analysis and design is the activity during which general testing objectives are transformed into tangible test conditions and test cases.

The test analysis and design activity has the following major tasks:
- Reviewing the test basis (such as requirements, software integrity level[1] (risk level), risk analysis reports, architecture, design, interface specifications)
- Evaluating testability of the test basis and test objects
- Identifying and prioritizing test conditions based on analysis of test items, the specification, behavior and structure of the software
- Designing and prioritizing high level test cases
- Identifying necessary test data to support the test conditions and test cases
- Designing the test environment setup and identifying any required infrastructure and tools
- Creating bi-directional traceability between test basis and test cases

[1] The degree to which software complies or must comply with a set of stakeholder-selected software and/or software-based system characteristics (e.g., software complexity, risk assessment, safety level, security level, desired performance, reliability, or cost) which are defined to reflect the importance of the software to its stakeholders.

1.4.3 Test Implementation and Execution (K1)

Test implementation and execution is the activity where test procedures or scripts are specified by combining the test cases in a particular order and including any other information needed for test execution, the environment is set up and the tests are run.

Test implementation and execution has the following major tasks:
- Finalizing, implementing and prioritizing test cases (including the identification of test data)
- Developing and prioritizing test procedures, creating test data and, optionally, preparing test harnesses and writing automated test scripts
- Creating test suites from the test procedures for efficient test execution
- Verifying that the test environment has been set up correctly
- Verifying and updating bi-directional traceability between the test basis and test cases
- Executing test procedures either manually or by using test execution tools, according to the planned sequence
- Logging the outcome of test execution and recording the identities and versions of the software under test, test tools and testware
- Comparing actual results with expected results
- Reporting discrepancies as incidents and analyzing them in order to establish their cause (e.g., a defect in the code, in specified test data, in the test document, or a mistake in the way the test was executed)
- Repeating test activities as a result of action taken for each discrepancy, for example, re-execution of a test that previously failed in order to confirm a fix (confirmation testing), execution of a corrected test and/or execution of tests in order to ensure that defects have not been introduced in unchanged areas of the software or that defect fixing did not uncover other defects (regression testing)

1.4.4 Evaluating Exit Criteria and Reporting (K1)

Evaluating exit criteria is the activity where test execution is assessed against the defined objectives. This should be done for each test level (see Section 2.2).

Evaluating exit criteria has the following major tasks:
- Checking test logs against the exit criteria specified in test planning
- Assessing if more tests are needed or if the exit criteria specified should be changed
- Writing a test summary report for stakeholders

1.4.5 Test Closure Activities (K1)

Test closure activities collect data from completed test activities to consolidate experience, testware, facts and numbers. Test closure activities occur at project milestones such as when a software system is released, a test project is completed (or cancelled), a milestone has been achieved, or a maintenance release has been completed.

Test closure activities include the following major tasks:
- Checking which planned deliverables have been delivered
- Closing incident reports or raising change records for any that remain open
- Documenting the acceptance of the system
- Finalizing and archiving testware, the test environment and the test infrastructure for later reuse
- Handing over the testware to the maintenance organization
- Analyzing lessons learned to determine changes needed for future releases and projects
- Using the information gathered to improve test maturity

1.5 The Psychology of Testing (K2) — *25 minutes*

Terms
Error guessing, independence

Background
The mindset to be used while testing and reviewing is different from that used while developing software. With the right mindset developers are able to test their own code, but separation of this responsibility to a tester is typically done to help focus effort and provide additional benefits, such as an independent view by trained and professional testing resources. Independent testing may be carried out at any level of testing.

A certain degree of independence (avoiding the author bias) often makes the tester more effective at finding defects and failures. Independence is not, however, a replacement for familiarity, and developers can efficiently find many defects in their own code. Several levels of independence can be defined as shown here from low to high:
- Tests designed by the person(s) who wrote the software under test (low level of independence)
- Tests designed by another person(s) (e.g., from the development team)
- Tests designed by a person(s) from a different organizational group (e.g., an independent test team) or test specialists (e.g., usability or performance test specialists)
- Tests designed by a person(s) from a different organization or company (i.e., outsourcing or certification by an external body)

People and projects are driven by objectives. People tend to align their plans with the objectives set by management and other stakeholders, for example, to find defects or to confirm that software meets its objectives. Therefore, it is important to clearly state the objectives of testing.

Identifying failures during testing may be perceived as criticism against the product and against the author. As a result, testing is often seen as a destructive activity, even though it is very constructive in the management of product risks. Looking for failures in a system requires curiosity, professional pessimism, a critical eye, attention to detail, good communication with development peers, and experience on which to base error guessing.

If errors, defects or failures are communicated in a constructive way, bad feelings between the testers and the analysts, designers and developers can be avoided. This applies to defects found during reviews as well as in testing.

The tester and test leader need good interpersonal skills to communicate factual information about defects, progress and risks in a constructive way. For the author of the software or document, defect information can help them improve their skills. Defects found and fixed during testing will save time and money later, and reduce risks.

Communication problems may occur, particularly if testers are seen only as messengers of unwanted news about defects. However, there are several ways to improve communication and relationships between testers and others:

- Start with collaboration rather than battles – remind everyone of the common goal of better quality systems
- Communicate findings on the product in a neutral, fact-focused way without criticizing the person who created it, for example, write objective and factual incident reports and review findings
- Try to understand how the other person feels and why they react as they do
- Confirm that the other person has understood what you have said and vice versa

1.6 Code of Ethics — *10 minutes*

Involvement in software testing enables individuals to learn confidential and privileged information. A code of ethics is necessary, among other reasons to ensure that the information is not put to inappropriate use. Recognizing the ACM and IEEE code of ethics for engineers, the ISTQB states the following code of ethics:

PUBLIC - Certified software testers shall act consistently with the public interest

CLIENT AND EMPLOYER - Certified software testers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest

PRODUCT - Certified software testers shall ensure that the deliverables they provide (on the products and systems they test) meet the highest professional standards possible

JUDGMENT- Certified software testers shall maintain integrity and independence in their professional judgment

MANAGEMENT - Certified software test managers and leaders shall subscribe to and promote an ethical approach to the management of software testing

PROFESSION - Certified software testers shall advance the integrity and reputation of the profession consistent with the public interest

COLLEAGUES - Certified software testers shall be fair to and supportive of their colleagues, and promote cooperation with software developers

SELF - Certified software testers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession

References

1.1.5 Black, 2001, Kaner, 2002
1.2 Beizer, 1990, Black, 2001, Myers, 1979
1.3 Beizer, 1990, Hetzel, 1988, Myers, 1979
1.4 Hetzel, 1988
1.4.5 Black, 2001, Craig, 2002
1.5 Black, 2001, Hetzel, 1988