might put the company out of business; if you tried to apply e-commerce approaches to safety-critical software, you could put lives in danger. So, the context of the testing influences how much testing we do and how the testing is done.

Another example is the way that testing is done in an Agile project as opposed to a sequential life cycle project. Every sprint in an Agile project includes testing of the functionality developed in that sprint; the testing is done by everyone on the Agile team (ideally) and the testing is done continually over the whole of development. In sequential life cycle projects, testing may be done more formally, documented in more detail and may be focused towards the end of the project.

Principle 7. Absence-of-errors is a fallacy

Throughout this section we have expounded the idea that a sequence of test activities, started early and targeting specific and diverse objectives and areas of the system, can effectively and efficiently find – and help a project team to remove – a large percentage of the defects. Surely that is all that is required to achieve project success?

Sadly, it is not. Many systems have been built that failed in user acceptance testing or in the marketplace, such as the initial launch of the US healthcare.gov website, which suffered from serious performance and web access problems.

Consider desktop computer operating systems. In the 1990s, as competition peaked for dominance of the PC operating system market, Unix and its variants had higher levels of quality than DOS and Windows. However, 25 years on, Windows dominates the desktop marketplace. One major reason is that Unix and its variants were too difficult for most users in the early 1990s.

Consider a system that perfectly conforms to its requirements (if that were possible), which has been tested thoroughly and all defects found have been fixed. Surely this would be a success, right? Wrong! If the requirements were flawed, we now have a perfectly working wrong system. Perhaps it is hard to use, as in the previous example. Perhaps the requirements missed some major features that users were expecting or needed to have. Perhaps this system is quite OK, but a competitor has come out with a competing system that is easier to use, includes the expected features and is cheaper. Our 'perfect' system is not looking so good after all, even though it has effectively 'no defects' in terms of 'conformance to requirements'.

1.4 TEST PROCESS

SYLLABUS LEARNING OBJECTIVES FOR 1.4 TEST PROCESS (K2)

FL-1.4.1 **Explain the impact of context on the test process (K2)**

FL-1.4.2 **Describe the test activities and respective tasks within the test process (K2)**

FL-1.4.3 **Differentiate the work products that support the test process (K2)**

FL-1.4.4 **Explain the value of maintaining traceability between the test basis and test work products (K2)**

In this section, we will describe the test process: tasks, activities and work products. We will talk about the influence of context on the test process and the importance of traceability.

In this section, there are a large number of Glossary keywords (19 in all): **coverage**, **test analysis**, **test basis**, **test case**, **test completion**, **test condition**, **test control**, **test data**, **test design**, **test execution**, **test execution schedule**, **test implementation**, **test monitoring**, **test oracle**, **test planning**, **test procedure**, **test suite**, **testware** and **traceability**.

In Section 1.1, we looked at the definition of testing, and identified misperceptions about testing, including that testing is not just test execution. Certainly, test execution is the most visible testing activity. However, effective and efficient testing requires test approaches that are properly planned and carried out, with tests designed and implemented to cover the proper areas of the system, executed in the right sequence and with their results reviewed regularly. This is a process, with tasks and activities that can be identified and need to be done, sometimes formally and other times very informally. In this section, we will look at the test process in detail.

There is no 'one size fits all' test process, but testing does need to include common sets of activities, or it may not achieve its objectives. An organization may have a test strategy where the test activities are specified, including how they are implemented and when they occur within the life cycle. Another organization may have a test strategy where test activities are not formally specified, but expertise about test activities is shared among team members informally. The 'right' test process for you is one that achieves your test objectives in the most efficient way. The best test process for you would not be the best for another organization (and vice versa).

Simply having a defined test strategy is not enough. One of our clients recently was a law firm that sued a company for a serious software failure. It turned out that while the company had a written test strategy, this strategy was not aligned with the testing best practices described in this book or the Syllabus. Further, upon close examination of their test work products, it was clear that they had not even carried out the strategy properly or completely. The company ended up paying a substantial penalty for their lack of quality. So, you must consider whether your actual test activities and tasks are sufficient.

1.4.1 Test process in context

As mentioned above, there is no one right test process that applies to everyone; each organization needs to adapt their test process depending on their context. The factors that influence the particular test process include the following (this list is not exhaustive):

- Software development life cycle model and project methodologies being used. An Agile project developing mobile apps will have quite a different test process to an organization producing medical devices such as pacemakers.
- Test levels and test types being considered; for example, a large complex project may have several types of integration testing, with a test process reflecting that complexity.
- Product and project risks (the lower the risks, the less formal the process needs to be, and vice versa).

- Business domain (e.g. mobile apps versus medical devices).
- Operational constraints, including:
  - budgets and resources
  - timescales
  - complexity
  - contractual and regulatory requirements.
- Organizational policies and practices.
- Required internal and external standards.

In the following sections, we will look in detail at the following topics:

- Test activities and tasks – the various things that testers do.
- Test work products – the things produced by and used by testers.
- Traceability between the test basis and test work products and why this is important.

First, one aspect to consider is **coverage**. Coverage is a partial measure of the thoroughness of testing. When examining the **test basis**, which is whatever the tests are being derived from (such as a requirement, user story, design or even code), a number of things can be identified as coverage items (we can tell whether or not we have tested them). For example, system level testing may want to ensure that every user story has been tested at least once; integration level testing may want to ensure that every communication path has been tested at least once; and, in component testing, developers may want to ensure that every code module, branch or statement has been tested at least once. When a test exercises the coverage item (user story, communication path or code element), then that item has been covered. Coverage is the percentage of coverage items that were exercised in a given test run.

Coverage The degree to which specified coverage items have been determined to have been exercised by a test suite expressed as a percentage.

Test basis The body of knowledge used as the basis for test analysis and design.

It is useful to know what you want to cover with your testing right from the start; coverage can act as a Key Performance Indicator (KPI) and help to measure the achievement of test objectives (if they are related to coverage).

In addition to the coverage items relating to specifications or code, we may also have environmental aspects that we want to cover. For example, tests for mobile apps may need to be tested on a number of mobile devices and configurations – these are also coverage items or coverage criteria. We could also consider coverage items of user personas, stakeholders or user success criteria. Measuring and reporting the coverage of these aspects can also give confidence to stakeholders that failures in operation would be less likely.

There is more about coverage in Section 4.3. Test processes are described in more detail in ISO/IEC/IEEE 29119-2 [2013].

1.4.2 Test activities and tasks

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

- Test planning.
- Test monitoring and control.
- Test analysis.
- Test design.

- Test implementation.
- Test execution.
- Test completion.

These activities appear to be logically sequential, in the sense that tasks within each activity often create the preconditions or precursor work products for tasks in subsequent activities. However, in many cases, the activities in the process may overlap or take place concurrently or iteratively, provided that these dependencies are fulfilled. Each group of activities consists of many individual tasks; these will vary for different projects or releases. For example, in Agile development, we have small iterations of software design, build and test that happen continuously, and planning is also a very dynamic activity throughout. If there are multiple teams, some teams may be doing test analysis while other teams are in the middle of test implementation, for example.

Note that this is 'a' test process, not 'the' test process. We have found that most of these activities, and many of the tasks within these activities, are carried out in some form or another on most successful test efforts. However, you should expect to have to tailor your test process, its main activities and the constituent tasks based on the organizational, project, process and product needs, constraints and other contextual realities. In sequential development, there will also be overlap, combination, concurrency or even omission of some tasks; this is why a test process is tailored for each project.

Test planning The activity of establishing or updating a test plan.

Test plan Documentation describing the test objectives to be achieved and the means and the schedule for achieving them, organized to coordinate testing activities.

(Note that we have included the definition of test plan here, even though it is not listed in the Syllabus as a term that you need to know for this chapter; otherwise the definition of test planning is not very informative.)

Test monitoring A test management activity that involves checking the status of testing activities, identifying any variances from the planned or expected status and reporting status to stakeholders.

Test control A test management task that deals with developing and applying a set of corrective actions to get a test project on track when monitoring shows a deviation from what was planned.

Test planning

Test planning involves defining the objectives of testing and the approach for meeting those objectives within project constraints and contexts. This includes deciding on suitable test techniques to use, deciding what tasks need to be done, formulating a test schedule and other things.

Metaphorically, you can think of test planning as similar to figuring out how to get from one place to another (without using your GPS – there is no GPS for testing). For small, simple and familiar projects, finding the route merely involves taking an existing map, highlighting the route and jotting down the specific directions. For large, complex or new projects, finding the route can involve a sophisticated process of creating a new map, exploring unknown territory and blazing a fresh trail.

We will discuss test planning in more detail in Section 5.2.

Test monitoring and control

To continue our metaphor, even with the best map and the clearest directions, getting from one place to another involves careful attention, watching the dashboard, minor (and sometimes major) course corrections, talking with our companions about the journey, looking ahead for trouble, tracking progress towards the ultimate destination and coping with finding an alternate route if the road we wanted is blocked. So, in **test monitoring**, we continuously compare actual progress against the plan, check on the progress of test activities and report the test status and any necessary deviations from the plan. In **test control**, we take whatever actions are necessary to meet the mission and objectives of the project, and/or adjust the plan.

Test monitoring is the ongoing comparison of actual progress against the test plan, using any test monitoring metrics that we have defined in the test plan. Test progress against the plan is reported to stakeholders in test progress reports or stakeholder meetings. One option that is often overlooked is that if things are going very wrong,

it may be time to stop the testing or even stop the project completely. In our driving analogy, once you find out that you are headed in completely the wrong direction, the best option is to stop and re-evaluate, not continue driving to the wrong place.

One way we can monitor test progress is by using exit criteria, also known as 'definition of done' in Agile development. For example, the exit criteria for test execution might include:

- Checking test results and logs against specified coverage criteria (we have not finished testing until we have tested what we planned to test).
- Assessing the level of component or system quality based on test results and logs (e.g. the number of defects found or ease of use).
- Assessing product risk and determining if more tests are needed to reduce the risk to an acceptable level.

We will discuss test planning, monitoring and control tasks in more detail in Chapter 5.

Test analysis

In **test analysis**, we analyze the test basis to identify testable features and define associated **test conditions**. Test analysis determines 'what to test', including measurable coverage criteria. We can say colloquially that the test basis is everything upon which we base our tests. The test basis can include requirements, user stories, design specifications, risk analysis reports, the system design and architecture, interface specifications and user expectations.

Test analysis The activity that identifies test conditions by analyzing the test basis.

Test condition (charter) An aspect of the test basis that is relevant in order to achieve specific test objectives. See also: exploratory testing.

In test analysis, we transform the more general testing objectives defined in the test plan into tangible test conditions. The way in which these are specifically documented depends on the needs of the testers, the expectations of the project team, any applicable regulations and other considerations.

Test analysis includes the following major activities and tasks:

- Analyze the test basis appropriate to the test level being considered. Examples of a test basis include:
  - Requirement specifications, for example, business requirements, functional requirements, system requirements, user stories, epics, use cases or similar work products that specify desired functional and non-functional component or system behaviour. These specifications say what the component or system should do and are the source of tests to assess functionality as well as non-functional aspects such as performance or usability.
  - Design and implementation information, such as system or software architecture diagrams or documents, design specifications, call flows, modelling diagrams (for example, UML or entity-relationship diagrams), interface specifications or similar work products that specify component or system structure. Structures for implemented systems or components can be a useful source of coverage criteria to ensure that sufficient testing has been done on those structures.
  - The implementation of the component or system itself, including code, database metadata and queries, and interfaces. Use all information about any aspect of the system to help identify what should be tested.
  - Risk analysis reports, which may consider functional, non-functional and structural aspects of the component or system. Testing should be more thorough in the areas of highest risk, so more test conditions should be identified in the highest-risk areas.

- Evaluate the test basis and test items to identify various types of defects that might occur (typically done by reviews), such as:
  - ambiguities
  - omissions
  - inconsistencies
  - inaccuracies
  - contradictions
  - superfluous statements.
- Identify features and sets of features to be tested.
- Identify and prioritize test conditions for each feature, based on analysis of the test basis, and considering functional, non-functional and structural characteristics, other business and technical factors, and levels of risks.
- Capture bi-directional traceability between each element of the test basis and the associated test conditions. This traceability should be bi-directional (we can trace in both forward and backward directions) so that we can check which test basis elements go with which test conditions (and vice versa) and determine the degree of coverage of the test basis by the test conditions. See Sections 1.4.3 and 1.4.4 for more on traceability. Traceability is also very important for maintenance testing, as we will discuss in Chapter 2, Section 2.4.

How are the test conditions actually identified from a test basis? The test techniques, which are described in Chapter 4, are used to identify test conditions. Black-box techniques identify functional and non-functional test conditions, white-box techniques identify structural test conditions and experience-based techniques can identify other important test conditions. Using techniques helps to reduce the likelihood of missing important conditions and helps to define more precise and accurate test conditions.

Sometimes the test conditions identified can be used as test objectives for a test charter. In exploratory testing, an experience-based technique (see Chapter 4, Section 4.4.2), test charters are used as goals for the testing that will be carried out in an exploratory way – that is, test design, execution and learning in parallel. When these test objectives are traceable to the test basis, the coverage of those test conditions can be measured.

One of the most beneficial side effects of identifying what to test in test analysis is that you will find defects; for example, inconsistencies in requirements, contradictory statements between different documents, missing requirements (such as no 'otherwise' for a selection of options) or descriptions that do not make sense. Rather than being a problem, this is a great opportunity to remove these defects before development goes any further. This verification (and validation) of specifications is particularly important if no other review processes for the test basis documents are in place.

Test analysis can also help to validate whether the requirements properly capture customer, user and other stakeholder needs. For example, techniques such as behaviour-driven development (BDD) and acceptance test-driven development (ATDD) both involve generating test conditions (and test cases) from user stories. BDD focuses on the behaviour of the system and ATDD focuses on the user view of the system, and both techniques involve defining acceptance criteria. Since these acceptance criteria are produced before coding, they also verify and validate the user stories and the acceptance criteria. More about this is found in the ISTQB Foundation Level Agile Tester Extension qualification.

Test design

Test analysis addresses 'what to test' and **test design** addresses the question 'how to test'; that is, what specific inputs and data are needed in order to exercise the software for a particular test condition. In test design, test conditions are elaborated (at a high level) in **test cases**, sets of test cases and other testware. Test analysis identifies general 'things' to test, and test design makes these general things specific for the component or system that we are testing.

Test design includes the following major activities:

- Design and prioritize test cases and sets of test cases.
- Identify the necessary **test data** to support the test conditions and test cases as they are identified and designed.
- Design the test environment, including set-up, and identify any required infrastructure and tools.
- Capture bi-directional traceability between the test basis, test conditions, test cases and test procedures (see also Section 1.4.4).

As with the identification of test conditions, test techniques are used to derive or elaborate test cases from the test conditions. These are described in Chapter 4, where test analysis and test design are discussed in more detail.

Just as in test analysis, test design can also identify defects – in the test basis and in the existing test conditions. Because test design is a deeper level of detail, some defects that were not obvious when looking at test basis at a high level, may become clear when deciding exactly what values to assign to test cases. For example, a test condition might be to check the boundary values of an input field, but when determining the exact values, we realize that a maximum value has not been specified in the test basis. Identifying defects at this point is a good thing because if they are fixed now, they will not cause problems later.

Which of these specific tasks applies to a particular project depends on various contextual issues relevant to the project, and these are discussed further in Chapter 5.

Test design The activity of deriving and specifying test cases from test conditions.

Test case A set of preconditions, inputs, actions (where applicable), expected results and postconditions, developed based on test conditions.

Test data Data created or selected to satisfy the execution preconditions and inputs to execute one or more test cases.

Test implementation

In **test implementation**, we specify **test procedures** (or test scripts). This involves combining the test cases in a particular order, as well as including any other information needed for test execution. Test implementation also involves setting up the test environment and anything else that needs to be done to prepare for test execution, such as creating testware. Test design asked 'how to test', and test implementation asks 'do we now have everything in place to run the tests?'

Test implementation includes the following major activities:

- Develop and prioritize the test procedures and, potentially, create automated test scripts.
- Create **test suites** from the test procedures and automated test scripts (if any). See Chapter 6 for test automation.
- Arrange the test suites within a **test execution schedule** in a way that results in efficient test execution (see Chapter 5, Section 5.2.4).
- Build the test environment (possibly including test harnesses, service virtualization, simulators and other infrastructure items) and verify that everything needed has been set up correctly.

Test implementation The activity that prepares the testware needed for test execution based on test analysis and design.

Test procedure A sequence of test cases in execution order, and any associated actions that may be required to set up the initial preconditions and any wrap-up activities post execution.

Test suite (test case suite, test set) A set of test cases or test procedures to be executed in a specific test cycle.

Test execution schedule A schedule for the execution of test suites within a test cycle.

- Prepare test data and ensure that it is properly loaded in the test environment (including inputs, data resident in databases and other data repositories, and system configuration data).
- Verify and update the bi-directional traceability between the test basis, test conditions, test cases, test procedures and test suites (see also Section 1.4.4).

Ideally, all of these tasks are completed before test execution begins, because otherwise precious, limited test execution time can be lost on these types of preparatory tasks. One of our clients reported losing as much as 25% of the test execution period to what they called 'environmental shakedown', which turned out to consist almost entirely of test implementation activities that could have been completed before the software was delivered.

Note that although we have discussed test design and test implementation as separate activities, in practice they are often combined and done together.

Not only are test design and implementation combined, but many test activities may be combined and carried out concurrently. For example, in exploratory testing (see Chapter 4, Section 4.4.2), test analysis, test design, test implementation and test execution are done in an interactive way throughout an exploratory test session.

Test execution

Test execution The process of running a test on the component or system under test, producing actual result(s).

Testware Work products produced during the test process for use in planning, designing, executing, evaluating and reporting on testing.

In **test execution**, the test suites that have been assembled in test implementation are run, according to the test execution schedule.

Test execution includes the following major activities:

- Record the identities and versions of all of the test items (parts of the test object to be tested), test objects (system or component to be tested), test tools and other **testware**.
- Execute the tests either manually or by using an automated test execution tool, according to the planned sequence.
- Compare actual results with expected results, observing where the actual and expected results differ. These differences may be the result of defects, but at this point we do not know, so we refer to them as anomalies.
- Analyze the anomalies in order to establish their likely causes. Failures may occur due to defects in the code or they may be false-positives. (A false-positive is where a defect is reported when there is no defect.) A failure may also be due to a test defect, such as defects in specified test data, in a test document or the test environment, or simply due to a mistake in the way the test was executed.
- Report defects based on the failures observed (see Chapter 5, Section 5.6). A failure due to a defect in the code means that we can write a defect report. Some organizations track test defects (i.e. defects in the tests themselves), while others do not.
- Log the outcome of test execution (e.g. pass, fail or blocked). This includes not only the anomalies observed and the pass/fail status of the test cases, but also the identities and versions of the software under test, test tools and testware.

- As necessary, repeat test activities when actions are taken to resolve discrepancies. For example, we might need to re-run a test that previously failed in order to confirm a fix (confirmation testing). We might need to run an updated test. We might also need to run additional, previously executed tests to see whether defects have been introduced in unchanged areas of the software or to see whether a fixed defect now makes another defect apparent (regression testing).
- Verify and update the bi-directional traceability between the test basis, test conditions, test cases, test procedures and test results.

As before, which of these specific tasks applies to a particular project depends on various contextual issues relevant to the project; these are discussed further in Chapter 5.

Test completion

Test completion activities collect data from completed test activities to consolidate experience, testware and any other relevant information. Test completion activities should occur at major project milestones. These can include when a software system is released, when a test project is completed (or cancelled), when an Agile project iteration is finished (e.g. as part of a retrospective meeting), when a test level has been completed or when a maintenance release has been completed. The specific milestones that involve test completion activities should be specified in the test plan.

Test completion The activity that makes test assets available for later use, leaves test environments in a satisfactory condition and communicates the results of testing to relevant stakeholders.

Test completion includes the following major activities:

- Check whether all defect reports are closed, entering change requests or product backlog items for any defects that remain unresolved at the end of test execution.
- Create a test summary report to be communicated to stakeholders.
- Finalize and archive the test environment, the test data, the test infrastructure and other testware for later reuse.
- Hand over the testware to the maintenance teams, other project teams, and/or other stakeholders who could benefit from its use.
- Analyze lessons learned from completed test activities to determine changes needed for future iterations, releases and projects (i.e. perform a retrospective).
- Use the information gathered to improve test process maturity, especially as an input to test planning for future projects.

The degree and extent to which test completion activities occur, and which specific test completion activities do occur, depends on various contextual issues relevant to the project, which are discussed further in Chapter 5.

1.4.3 Test work products

'Work products' is the generic name given to any form of documentation, informal communication or artefact that is used in testing (or indeed in development). When testing is more formal, the majority of work products may be written documentation; when testing is very informal, the corresponding work product may just be a scrap of paper, or a note in someone's mobile phone. 'Work product' is a general term covering any type of information needed to do the work (testing or development).