- **Transfer:** Convince some other member of the team or project stakeholder to reduce the likelihood or accept the impact of the risk.
- **Ignore:** Do nothing about the risk, which is usually a smart option only when there's little that can be done or when the likelihood and impact are low.

There is another typical risk management option, buying insurance, which is not usually pursued for project or product risks on software projects, though it is not unheard of.

Risk management activities include:

- Identifying and analyzing (and re-evaluating on a regular basis) what can go wrong.
- Determining the priority of risks, which risks are the most important to deal with.
- Implementing actions to mitigate those risks, to reduce their likelihood or impact or both.
- Making contingency plans to deal with risks if they do happen.

We can deal with test-related risks to the project and product by applying some straightforward, structured risk management techniques. The first step is to assess or analyze risks early in the project. Like a big ocean liner, projects, especially large projects, require steering well before the iceberg is in plain sight. By using a test plan, you can remind yourself to consider and manage risks during the planning activity.

Risk-based testing

Risk-based testing is the idea that we can organize our testing efforts in a way that reduces the residual level of product risk when the system is delivered. Risk-based testing uses risk to prioritize and emphasize the appropriate tests during test execution, but it is about more than that. Risk-based testing starts early in the project, identifying risks to system quality and using that knowledge of risk to guide testing planning, specification, preparation and execution. Risk-based testing involves both mitigation (testing to provide opportunities to reduce the likelihood of defects, especially high-impact defects) and contingency (testing to identify work-arounds to make the defects that do get past us less painful). Risk-based testing also involves measuring how well we are doing at finding and removing defects in critical areas, as was shown in Table 5.1. Risk-based testing can also involve using risk analysis to identify proactive opportunities to remove or prevent defects through non-testing activities and to help us select which test activities to perform.

Risk-based testing
Testing in which the management, selection, prioritization and use of testing activities and resources are based on corresponding risk types and risk levels.

Mature test organizations use testing to reduce the risk associated with delivering the software to an acceptable level [Beizer 1990, Hetzel 1988]. In the middle of the 1990s, a number of testers, including us, started to explore various techniques for risk-based testing. In doing so, we adapted well-accepted risk management concepts to software testing. Applying and refining risk assessment and management techniques are discussed in Black [2004] and Black [2009]. For two alternative views, see Chapter 11 of Pol, Teunissen and van Veenendaal [2002] and Chapter 2 of Craig and Jaskiel [2002]. The origin of the risk-based testing concept can be found in Chapter 1 of Beizer [1990] and Chapter 2 of Hetzel [1988].

Risk analysis

Risk-based testing starts with product risk analysis. One technique for risk analysis is a close reading of the requirements specification, user stories, design specifications, user documentation and other items. Another technique is brainstorming with many of the project stakeholders. Another is a sequence of one-to-one or small group sessions with the business and technology experts in the company. Some people use all these techniques when they can. To us, a team-based approach that involves the key stakeholders and experts is preferable to a purely document-based approach, as team approaches draw on the knowledge, wisdom and insight of the entire team to determine what to test and how much.

While you could perform the risk analysis by asking, 'What should we worry about?' usually more structure is required to avoid missing things. One way to provide that structure is to look for specific risks in particular product risk categories. You could consider risks in the areas of functionality, localization, usability, reliability, performance and supportability. Alternatively, you could use the quality characteristics and sub-characteristics from ISO/IEC 25010 [2011] (introduced in Chapter 2), as each sub-characteristic that matters is subject to risks that the system might have troubles in that area. You might have a checklist of typical or past risks that should be considered. You might also want to review the tests that failed and the bugs that you found in a previous release or a similar product. These lists and reflections serve to jog the memory, forcing you to think about risks of particular types, as well as helping you structure the documentation of the product risks.

When we talk about specific risks, we mean a particular kind of defect or failure that might occur. For example, if you were testing the calculator utility that is bundled with Microsoft Windows, you might identify incorrect calculation as a specific risk within the category of functionality. However, this is too broad. Consider incorrect addition. This is a high-impact kind of defect, as everyone who uses the calculator will see it. It is unlikely, since addition is not a complex algorithm. Contrast that with an incorrect sine calculation. This is a low-impact kind of defect, since few people use the sine function on the Windows calculator. It is more likely to have a defect, though, since sine functions are hard to calculate.

It's worth making the point here that in early risk analysis, we are making educated guesses. Some of those guesses will be wrong. Make sure that you plan to re-assess and adjust your risks at regular intervals in the project and make appropriate course corrections to the testing or the project itself.

One common problem people have when organizations first adopt risk-based testing is a tendency to be excessively alarmed by some of the risks once they are clearly articulated. Don't confuse impact with likelihood or vice versa. You should manage risks appropriately, based on likelihood and impact. Triage the risks by understanding how much of your overall effort can be spent dealing with them.

It's very important to maintain a sense of perspective, a focus on the point of the exercise. As with life, the goal of risk-based testing should not be, but cannot practically be, a risk-free project. What we can accomplish with risk-based testing is the marriage of testing with best practices in risk management to achieve a project outcome that balances risks with quality, features, budget and schedule.

Assigning a risk level

After identifying the risk items, you and, if applicable, the stakeholders, should review the list to assign the likelihood of problems and the impact of problems associated with each one. There are many ways to go about this assignment of likelihood

and impact. You can do this with all the stakeholders at once. You can have the business people determine impact and the technical people determine likelihood, and then merge the determinations. Either way, the reason for identifying risks first and then assessing their level, is that the risks are relative to each other.

The scales used to rate likelihood and impact vary. Some people rate them high, medium and low. Some use a 1–10 scale. The problem with a 1–10 scale is that it is often difficult to tell a 2 from a 3 or a 7 from an 8, unless the differences between each rating are clearly defined. A five-point scale (very high, high, medium, low and very low) tends to work well.

Given two classifications of risk levels, likelihood and impact, we have a problem. We need a single, aggregate risk rating to guide our testing effort. As with rating scales, practices vary. One approach is to convert each risk classification into a number and then either multiply (or add) the numbers to calculate a risk priority number. For example, suppose a particular risk has a high likelihood and a medium impact. The risk priority number would then be 6 (2 times 3).

Mitigation options

Armed with a risk priority number, we can now decide on the various risk mitigation options available to us. Do we use formal training for developers or analysts, rely on cross-training and reviews or assume they know enough? Do we perform extensive testing, cursory testing or no testing at all? Should we ensure unit testing and system testing coverage of this risk? These options and more are available to us.

As you go through this process, make sure you capture the key information in a document. We are not fond of excessive documentation but this quantity of information simply cannot be managed in your head. We recommend a lightweight table like the one shown in Table 5.2, which we usually capture in a spreadsheet.

TABLE 5.2 A risk analysis template

Product risk	Likelihood	Impact	Risk priority number	Mitigation
Risk category 1				
Risk 1				
Risk 2				
Risk n				

Concluding thoughts on risk

To summarize, the results of product risk analysis are used for the following purposes:

- To determine the test techniques to be used.
- To determine the particular levels and types of testing to be performed (for example functional testing, security testing, performance testing).
- To determine the extent of testing to be carried out for the different levels and types of testing.

- To prioritize testing in order to find the most critical defects as early as possible.
- To determine whether any activities in addition to testing could be employed to reduce risk, such as providing training in design and testing to inexperienced developers.

Let's finish this section with two quick tips about product risk analysis. First, remember to consider both likelihood and impact. While it might make you feel like a hero to find lots of defects, testing is also about building confidence in key functions. We need to test the things that probably will not break, but would be catastrophic if they did.

Second, we re-iterate that in early risk analysis, we are making educated guesses. Make sure that you follow up and revisit the risk analysis at key project milestones. For example, if you are following a V-model, you might perform the initial analysis during the requirements phase, then review and revise it at the end of the design and implementation phases, as well as prior to starting unit test, integration test and system test. We also recommend revisiting the risk analysis during testing. You might find you have discovered new risks or found that some risks were not as risky as you thought and increased your confidence in the risk analysis. In Agile development, revisiting the risk analysis at the start of every sprint is a good idea.

5.6 DEFECT MANAGEMENT

SYLLABUS LEARNING OBJECTIVES FOR 5.6 DEFECT MANAGEMENT (K3)

FL-5.6.1 Write a defect report covering defects found during testing (K3)

Let's wind down this chapter on test management with an important subject: how we can document and manage the defects that occur during testing. One of the objectives of testing is to find defects, which reveal themselves as discrepancies between actual and expected outcomes. When we observe a defect, we need to log the details. Proper test management involves establishing appropriate actions to investigate and dispose of defects. This includes tracking defects from initial discovery and classification through to resolution, confirmation testing and ultimate disposition, with the defects following a clearly established set of rules and processes for defect management and classification. We'll look at what topics we should cover when reporting defects. At the end of this section, you will be ready to write a thorough defect report.

The only Syllabus term in this section is **defect management**.

What are defect reports for?

When running a test, you might observe actual results that vary from expected results. This is not a bad thing – one of the major goals of testing is to find problems. Different organizations have different names to describe such situations. Commonly, they are called incidents, bugs, defects, problems or issues.

To be precise, we sometimes draw a distinction between an incident or anomaly on the one hand and defects or bugs on the other. An incident or anomaly is any situation

where the system exhibits questionable behaviour, and a defect is some problem, imperfection or deficiency in the item we are testing.

Causes of defects include misconfiguration or failure of the test environment, corrupted test data, bad tests, incorrect expected results and tester mistakes. In some cases, the policy is to classify as a defect anything that arises from a test design, the test environment or anything else which is under formal configuration management. Bear in mind the possibility that a questionable behaviour is not necessarily a true defect. We log these defects so that we have a record of what we observed and can follow it up and track what is done to correct it, whether or not it turns out to be a problem in the work product we are testing or something else. This is **defect management**.

Defect management
The process of recognizing and recording defects, classifying them, investigating them, taking action to resolve them and disposing of them when resolved.

While it is most common to find defect logging or defect reporting processes and tools in use during formal, independent testing, you can also log, report, track and manage defects found during development and reviews. In fact, this is a good idea, because it gives useful information on the extent to which early (and cheaper) defect detection and removal activities are happening.

Of course, we also need some way of reporting, tracking and managing defects that occur in the field or after deployment of the system. While many of these defects will be user error or some other behaviour not related to a problem in system behaviour, some percentage of defects do escape from quality assurance and testing activities. The defect detection percentage (DDP), which compares field defects with test defects, can be a useful metric of the effectiveness of the test process.

Here is an example of a DDP formula that would apply for calculating DDP for the last level of testing prior to release to the field:

$$\text{DDP} = \frac{\text{defects (testers)}}{\text{defects (testers)} + \text{defects (field)}}$$

It is most common to find defects reported against the code or the system itself. However, we have also seen cases where defects are reported against requirements and design specifications, user and operator guides and tests. Often, it aids the effectiveness and efficiency of reporting, tracking and managing defects when the defect tracking tool provides an ability to vary some of the information captured depending on what the defect was reported against.

In some projects, a very large number of defects are found. Even on smaller projects where 100 or fewer defects are found, you can easily lose track of them unless you have a process for reporting, classifying, assigning and managing the defects from discovery to final resolution.

Objectives for defect reports

A defect report contains a description of the misbehaviour that was observed and classification of that misbehaviour. As with any written communication, it helps to have clear goals in mind when writing. Typical objectives for such reports include the following:

- To provide developers, managers and others with detailed information about the behaviour observed (the adverse event), that is, the defect. This will enable them to identify specific effects, to isolate the problem with a minimal reproducing test, and to correct the defect(s) as needed or to otherwise resolve the problem.

- To support test managers in the analysis of trends in aggregate defect data, either for understanding more about a particular set of problems or tests, or for understanding and reporting the overall level of system quality. This will enable them to track the quality of the work product and the impact on the testing. For example, if a lot of defects are being reported, the testers will do less testing, since their time is taken with writing defect reports. This also means that more confirmation tests will be needed.
- To enable defect reports to be analyzed over a project, and even across projects, to give information and ideas that can lead to development and test process improvements.

When writing a defect report, it helps to have the readers in mind, too. The developers need the information in the report to find and fix the defects. Before that happens, though, the defects should be reviewed and prioritized so that scarce testing and developer resources are spent fixing and confirmation testing the most important defects. Since some defects may be deferred – perhaps to be fixed later or perhaps, ultimately, not to be fixed at all – we should include work-arounds and other helpful information for help-desk or technical support teams. Finally, testers often need to know what their colleagues are finding so that they can watch for similar behaviour elsewhere and avoid trying to run tests that will be blocked.

How to write a good defect report: some tips

A good defect report is a technical document. In addition to being clear for its goals and audience, any good report grows out of a careful approach to researching and writing the report. We have some rules of thumb that can help you write a better defect report.

Use a careful, attentive approach to running your tests. You never know when you are going to find a problem. If you are pounding on the keyboard while gossiping with office mates or daydreaming about a movie you just saw, you might not notice strange behaviours. Even if you see the defect, how much do you really know about it? What can you write in your defect report?

Intermittent or sporadic symptoms are a fact of life for some defects and it is always discouraging to have a defect report bounced back as irreproducible. It's a good idea to try to reproduce symptoms when you see them. We have found three times to be a good rule of thumb. If a defect has intermittent symptoms, we would still report it, but we would be sure to include as much information as possible, especially how many times we tried to reproduce it and how many times it did in fact occur.

You should also try to isolate the defect by making carefully chosen changes to the steps used to reproduce it. In isolating the defect, you help guide the developer to the problematic part of the system. You also increase your own knowledge of how the system works, and how it fails.

Think outside the box. Some test cases focus on boundary conditions, which may make it appear that a defect is not likely to happen frequently in practice. We have found that it's a good idea to look for more generalized conditions that cause the failure to occur, rather than simply relying on the test case. This helps prevent the infamous defect report rejoinder, 'No real user is ever going to do that'. It also cuts down on the number of duplicate reports that get filed.

As there is often a lot of testing going on with the system during a test period, there are lots of other test results available. Comparing an observed problem against

other test results and known defects is a good way to find and document additional information that the developer is likely to find very useful. For example, you might check for similar symptoms observed with other defects, the same symptom observed with defects that were fixed in previous versions or similar (or different) results seen in tests that cover similar parts of the system.

Many readers of defect reports, managers especially, will need to understand the priority and severity of the defect. The impact of the problem on the users, customers and other stakeholders is important. Most defect tracking systems have a title or summary field and that field should mention the impact, too.

Choice of words definitely matters in defect reports. You should be clear and unambiguous. You should also be neutral, fact-focused and impartial, keeping in mind the testing-related interpersonal issues discussed in Chapter 1 and earlier in this chapter. Finally, keeping the report concise helps keep people's attention and avoids the problem of losing them in the details.

As a last rule of thumb for defect reports, we recommend that you use a review process for all reports filed. It works if you have the lead tester review reports and we have also allowed testers, at least experienced ones, to review other testers' reports. Reviews are proven quality assurance techniques and defect reports are important project deliverables.

What goes in a defect report?

A defect report describes some situation, behaviour or event that occurred during testing that was not as it should be, or that requires further investigation. In many cases, a defect report consists of one or two screens full of information gathered by a defect tracking tool and stored in a database.

A defect report filed during dynamic testing typically includes the following:

- An identifier.
- A title and short summary of the defect being reported.
- Date of the defect report, issuing organization, the date and time of the failure, the name of the tester, that is, the author of the defect report (and perhaps the reviewer of the test).
- Identification of the test item (configuration item being tested), the test environment and any additional information about the configuration of the software, system or environment.
- The development life cycle activity(s) or sprint in which the defect was observed.
- A description of the defect to enable reproduction and resolution, such as the steps to reproduce and the isolation steps tried.
- The expected and actual results of the test.
- The scope or degree of impact (severity) of the defect on the interests of stakeholder(s).
- Urgency/priority to fix.
- State of the defect report (for example open, deferred, duplicate, waiting to be fixed, awaiting confirmation testing, re-opened, closed).
- Conclusions, recommendations and approvals.
- Global issues, such as other areas that may be affected by a change resulting from the defect.

- Change history, such as the sequence of actions taken by project team members with respect to the defect, to isolate, repair and confirm it as fixed. These fields should mention the inputs given and outputs observed, the different ways you could – and could not – make the problem recur, and the impact.
- References, including the test case that revealed the problem, and references to specifications or other work products that provide information about correct behaviour.

Sometimes testers classify the scope, severity and priority of the defect, though sometimes managers or a bug triage committee handle that role.

If you are using a defect management tool, some of the information will be automatically generated, such as the time and date of the report, the ID number, the report author and the initial state (open).

An example defect report can be found in ISO/IEC/IEEE 29119-3 [2013].

As the defect is managed to resolution, managers might assign a level of priority to the report. The change control board or bug triage committee might document the risks, costs, opportunities and benefits associated with fixing or not fixing the defect. The developer, when fixing the defect, can capture the root cause, the time or development stage where it was introduced and the time or testing activity that identified it, and removed it (which may be different to when it was identified).

After the defect has been resolved, managers, developers or others may want to capture conclusions and recommendations. Throughout the life cycle of the defect report, from discovery to resolution, the defect tracking system should allow each person who works on the defect report to enter status and history information.

What happens to defect reports after you file them?

As we mentioned earlier, defect reports are managed through a life cycle from discovery to resolution. The defect report life cycle is often shown as a state transition diagram (see Figure 5.3). While your defect tracking system may use a different life cycle, let's take this one as an example to illustrate how a defect report life cycle might work.

In the defect report life cycle shown in Figure 5.3, all defect reports move through a series of clearly identified states after being reported. Some of these state transitions occur when a member of the project team completes some assigned task related to closing a defect report. Some of these state transitions occur when the project team decides not to repair a defect during this project, leading to the deferral of the defect report. Some of these state transitions occur when a defect report is poorly written or describes behaviour which is actually correct, leading to the rejection of that report.

Let's focus on the path taken by defect reports which are ultimately fixed. After a defect is reported, a peer tester or test manager reviews the report. If successful in the review, the defect report becomes opened, so now the project team must decide whether or not to repair the defect. If the defect is to be repaired, a developer is assigned to repair it.

Once the developer believes the repairs are complete, the defect report returns to the tester for confirmation testing. If the confirmation test fails, the defect report is re-opened and then re-assigned. Once the tester confirms a good repair, the defect report is closed. No further work remains to be done on this defect.

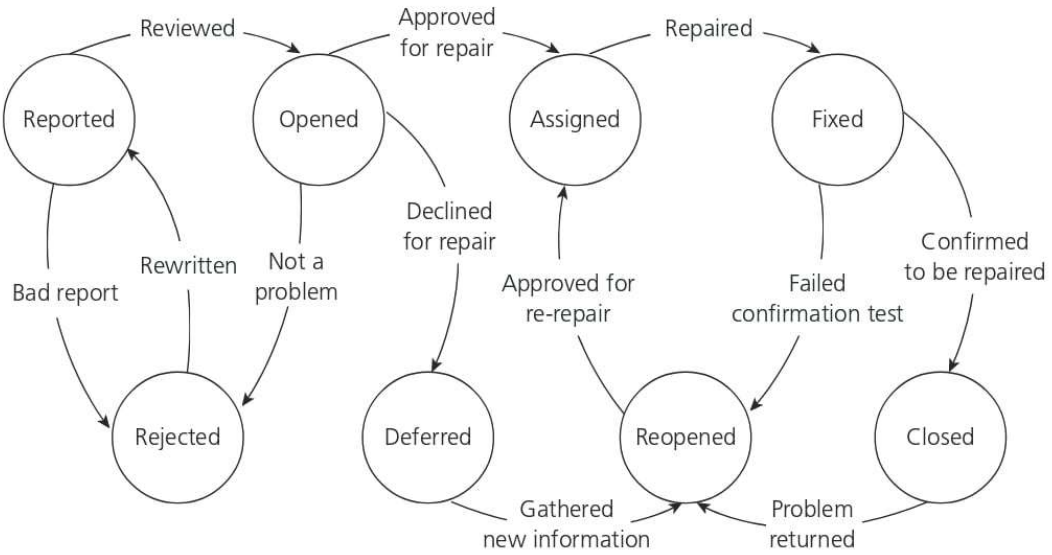

FIGURE 5.3 Defect report life cycle

In any state other than rejected, deferred or closed, further work is required on the defect prior to the end of this project. In such a state, the defect report has a clearly identified owner. The owner is responsible for transitioning the defect into an allowed subsequent state. The arrows in the diagram show these allowed transitions.

In a rejected, deferred or closed state, the defect report will not be assigned to an owner. However, certain real world events can cause a defect report to change state even if no active work is occurring on the defect report. Examples include the recurrence of a failure associated with a closed defect report and the discovery of a more serious failure associated with a deferred defect report.

Ideally, only the owner can transition the defect report from the current state to the next state and ideally the owner can only transition the defect report to an allowed next state. Most defect tracking systems support and enforce the defect life cycle and life cycle rules. Good defect tracking systems allow you to customize the set of states, the owners and the transitions allowed to match your actual workflows. And, while a good defect tracking system is helpful, the actual defect workflow should be monitored and supported by project and company management.

CHAPTER REVIEW

Let's review what you have learned in this chapter.

From Section 5.1, you should now be able to explain the basic ideas of test organization. You should know why independent testing is important, but also be able to analyze the potential benefits and problems associated with independent test teams. You should be able to recall the tasks that a tester and a test manager will carry out. You should know the Glossary terms **tester** and **test manager**.

From Section 5.2, you should now understand the fundamentals of test planning and estimation. You should know the reasons for writing test plans and be able to explain how test plans relate to projects, test levels, test objectives and test activities. You should be able to write a test execution schedule based on priority and dependencies. You should be able to explain the justification behind various entry and exit criteria that might relate to projects, test levels and or test objectives. You should be able to distinguish the purpose and content of test plans from that of test design specifications, test cases and test procedures. You should know the factors that affect the effort involved in testing, including especially test strategies (and approaches) and how they affect testing. You should be able to explain how metrics, expertise and negotiation are used for estimating. You should know the Glossary terms **entry criteria**, **exit criteria**, **test approach**, **test estimation**, **test plan**, **test planning** and **test strategy**.

From Section 5.3, you should be able to explain the essentials of test progress monitoring and control. You should know the common metrics that are captured, logged and used for monitoring, as well as ways to present these metrics. You should be able to explain a typical test progress report and test summary report. You should know the Glossary terms **test control**, **test monitoring**, **test progress report** and **test summary report**.

From Section 5.4, you should now understand the basics of configuration management that relate to testing, and how it helps us do our testing work better. You should know the Glossary term **configuration management**.

From Section 5.5, you should now be able to explain how risk is related to testing. You should know that a risk is a potential undesirable or negative outcome and that most of the risks we are interested in relate to the achievement of project objectives. You should know about likelihood and impact as factors that determine the importance of a risk. You should be able to compare and contrast risks to the product (and its quality) and risks to the project itself and know typical risks to the product and project. You should be able to describe how to use risk analysis and risk management for testing and test planning. You should know the Glossary terms **product risk**, **project risk**, **risk**, **risk level** and **risk-based testing**.

From Section 5.6, you should now understand defect logging and be able to use defect management on your projects. You should know the content of a defect report, be able to write a high quality report based on test results and manage that report through its life cycle. You should know the Glossary term **defect management**.

SAMPLE EXAM QUESTIONS

Question 1 What is a potential drawback of independent testing?

a. Independent testing is likely to find the same defects as developers.
b. Developers may lose a sense of responsibility for quality.
c. Independent testing may find more defects.
d. Independent testers may be too familiar with the system.

Question 2 Which of the following is among the typical tasks of a test manager?

a. Develop system requirements, design specifications and usage models.
b. Handle all test automation duties.
c. Keep tests and test coverage hidden from developers.
d. Gather and report test progress metrics.

Question 3 According to the ISTQB Glossary, what do we mean when we call someone a test manager?

a. A test manager manages a team of junior testers.
b. A test manager plans and controls testing activities.
c. A test manager sets up test environments.
d. A test manager creates a detailed test execution schedule.

Question 4 What is the primary difference between the test plan, the test design specification and the test procedure specification?

a. The test plan describes one or more levels of testing, the test design specification identifies the associated high-level test cases and a test procedure specification describes the actions for executing a test.
b. The test plan is for managers, the test design specification is for developers and the test procedure specification is for testers who are automating tests.
c. The test plan is the least thorough, the test procedure specification is the most thorough and the test design specification is midway between the two.
d. The test plan is finished in the first third of the project, the test design specification is finished in the middle third of the project and the test procedure specification is finished in the last third of the project.

Question 5 Which of the following factors is an influence on the test effort involved in most projects?

a. Geographical separation of tester and developers.
b. The departure of the test manager during the project.
c. The quality of the information used to develop the tests.
d. Unexpected long-term illness by a member of the project team.

Question 6 A business domain expert has given advice and guidance about the way the testing should be carried out and a technology expert has recommended using a checklist of common types of defect. Which two test strategies are being used?

a. Analytical and Process compliant.
b. Reactive and Methodical.
c. Directed and Methodical.
d. Regression-averse and Analytical.

Question 7 Consider the following exit criteria which might be found in a test plan:

I. No known customer-critical defects.
II. All interfaces between components tested.
III. 100% code coverage of all units.
IV. All specified requirements satisfied.
V. System functionality matches legacy system for all business rules.

Which of the following statements is true about whether these exit criteria belong in an acceptance test plan?

a. All statements belong in an acceptance test plan.
b. Only statement I belongs in an acceptance test plan.
c. Only statements I, II and V belong in an acceptance test plan.
d. Only statements I, IV and V belong in an acceptance test plan.

Question 8 According to the ISTQB Glossary, what is a test approach?

a. The implementation of the test strategy for a specific project.
b. Documentation that expresses the generic requirements for testing one or more projects within an organization providing detail on how testing is to be performed.
c. Documentation describing the test objectives to be achieved and the means and the schedule for achieving them, organized to coordinate testing activities.
d. The set of interrelated activities comprising of test planning, test monitoring and control, test analysis, test design, test implementation, test execution and test completion.

Question 9 Which of the following metrics would be most useful to monitor during test execution?

a. Percentage of test cases written.
b. Number of test environments remaining to be configured.
c. Number of defects found and fixed.
d. Percentage of requirements for which a test has been written.

Question 10 What would appear in a test progress report that would not be included in a test summary report?

a. Summary of testing performed.
b. The status of the test activities against the test plan.
c. Factors that have blocked or continue to block progress.
d. Reusable test products produced.

Question 11 In a defect report, the tester makes the following statement: 'The payment processing subsystem fails to accept payments from American Express cardholders, which is considered a must-work feature for this release.' This statement is likely to be found in which of the following sections?

a. Scope or degree of impact of the defect.
b. Identification of the test item.
c. Summary of the defect.
d. Description of the defect.

Question 12 During an early period of test execution, a defect is located, resolved and passed its confirmation test, but is seen again later during subsequent test execution. Which of the following is a testing-related aspect of configuration management that is most likely to have broken down?

a. Traceability.
b. Configuration item identification.
c. Configuration control.
d. Test documentation management.

Question 13 You are working as a tester on a project to develop a point-of-sales system for grocery stores and other similar retail outlets. Which of the following is a product risk for such a project?

a. The arrival of a more reliable competing product on the market.
b. Delivery of an incomplete test release to the first cycle of system test.
c. An excessively high number of defect fixes fail during confirmation testing.
d. Failure to accept allowed credit cards.

Question 14 What is a risk?

a. A bad thing that has happened to the product or the project.
b. A bad thing that might happen.
c. Something that costs a lot to put right.
d. Something that may happen in the future.

Question 15 You are writing a test plan and are currently completing a section on risk. Which of the following is most likely to be listed as a project risk?

a. Unexpected illness of a key team member.
b. Excessively slow transaction processing time.
c. Data corruption under network congestion.
d. Failure to handle a key use case.

Question 16 You and the project stakeholders develop a list of product risks and project risks during the planning stage of a project. What else should you do with those lists of risks during test planning?

a. Determine the extent of testing required for the product risks and the mitigation and contingency actions required for the project risks.

b. Obtain the resources needed to completely cover each product risk with tests and transfer responsibility for the project risks to the project manager.
c. Execute sufficient tests for the product risks, based on the likelihood and impact of each product risk and execute mitigation actions for all project risks.
d. No further risk management action is required at the test planning stage.

Question 17 The following are ways to estimate test effort:

1. Burndown charts.
2. Wideband Delphi.
3. Expert opinion.
4. Previous experience.
5. Planning poker.
6. Similar project data.

Which of these are an example of a metrics-based technique?

a. 3, 4 and 5.
b. 1, 5 and 6.
c. 2, 3 and 4.
d. 1, 2 and 5.

Question 18 In a defect report, the tester makes the following statement: 'At this point, I expect to receive an error message explaining the rejection of this invalid input and asking me to enter a valid input. Instead the system accepts the input, displays an hourglass for between one and five seconds and finally terminates abnormally, giving the message, "Unexpected data type: 15. Click to continue".' This statement is likely to be found in which of the following sections of a defect report?

a. Summary.
b. Impact.
c. Item pass/fail criteria.
d. Defect description.

Question 19 There are five tests to be executed. A high-priority test has been scheduled as the third in the sequence of five, rather than the first. Which of the following would NOT be a good reason for this?

a. The high priority test has a logical dependency on the two tests run before it.
b. All tests are the same priority and this is the most efficient sequence.
c. The tests scheduled before it are more likely to pass so this will take less time and give feedback more quickly.
d. The first three tests have the same priority so it does not matter what order they are run in.

Question 20 Exit criteria have been agreed at the start of the project, but it is now clear that they will not all be met before the release date next week. What should the test manager NOT say to those who want to release next week anyway?

a. These exit criteria were agreed for a very good reason, to avoid this situation. It is essential that all exit criteria are met before we release.
b. It is not my role to make that decision, so if you go ahead and release next week anyway, I will carefully monitor the effects and will report the results to high-level management.
c. Have all project stakeholders, business owners and the product owner been informed of the risks of releasing next week, and do they all understand and accept them? (Sign on this paper stating this.)
d. What are the most critical exit criteria, and can we get agreement from all stakeholders about which are absolutely necessary and which could be compromised this time?

EXERCISES

Test execution schedule exercise

You have been asked to develop a test execution schedule for the second release of a customer data management application. The first release only allowed basic customer records to be added. This second one contains 5 new features and 2 bug fixes, together with the new features' priorities (1 is highest) and the defect severity levels, in the table below.

TABLE 5.3 Exercise: Test execution schedule

Priority	Feature/Fix	Test order
1	Delete inactive customer record	
2	De-activate customer	
3	Edit extra data items	
4	Edit basic data	
5	Link customer records (for example in the same company)	
5	Provide extra data items when record added	

What would be the BEST test execution sequence for this release?

Defect report exercise

Assume you are testing a maintenance release which will add a new supported model to a browser-based application that allows car dealers to order custom-configured cars from the maker. You are working with a selected number of dealers who are performing beta testing. When they find problems, they send you an email describing the failure. You use that email to create a defect report in your defect management system.

You receive the following email from one of the dealers:

> *I entered an order for a Racinax 917X in midnight violet. During the upload, I got an 'unexpected return value' error message. I then checked the order and it was in the system. I think the internet connection might have gone down for a moment as we found out later that there was a power surge elsewhere in the building at around that time.*

Write a defect report based on this email and note any elements of such a report which might not be available based on this email alone.

Note that this scenario, including the name of the car model, is entirely fictitious.

EXERCISE SOLUTIONS

Test execution schedule exercise

Below we show the solution, with the tests listed in the test order. The justifications are shown below the table.

TABLE 5.4 Solution: Test execution schedule

Priority	Feature/Fix	Test order
2	De-activate customer	1
1	Delete inactive customer record	2
5	Provide extra data items when record added	3
3	Edit extra data items	4
4	Edit basic data	5
5	Link customer records (for example in the same company)	6

The highest priority is to Delete an inactive customer, but first we must have an inactive customer, so we run De-activate customer first. This is a functional dependency.

The next highest priority is to Edit extra data items, but again, we need to have some extra data items in order to do this, another functional dependency.

The final two are independent, so the priority 4 test is run before the priority 5 test.

Defect report exercise

Here is an example of a defect report:

Test defect report identifier: This would be assigned by the defect tracking tool.

Summary: System returns confusing error message if internet connectivity is interrupted.

Defect description:
Steps to reproduce

1. Entered an order for a Racinax 917X in midnight violet.
2. During the order upload, the system displayed an 'unexpected return value' error message.
3. Verified that, in spite of the error message, the order was in the system.

Suspected cause Given that a brief interruption in the internet connection may have occurred during order transmission due to the power surge, the suspected cause of the failure is a lack of robust handling of slow or unreliable internet connections.

Impact assessment The message in step 2 is not helpful to the user, because it gives the user no clues about what to do next. A user encountering the message described in step 2 might decide to re-enter the order, which in this case would result in a redundant order.
Some car dealerships will have unreliable internet connections, with the frequency of connection loss depending on location, wireless infrastructure available in the dealership, type of internet connection hardware used in the dealers' computers, and other such factors. Therefore we can expect that some number of defects such as this will occur in widespread use. (Indeed, this beta test defect proves that this is a likely event in the real world.) Since a careless or rushed dealer might decide to resubmit the order based on the error message, this will result in redundant orders, which will cause significant loss of profitability for the dealerships and the salespeople themselves.

This report addresses the inputs, the expected results, the actual results, the difference between the expected and actual results, and the procedure steps in which the tester identified the unexpected results. Here is some additional information needed to improve the report:

- The failure needs to be reproduced at least two more times. By doing the failure replication in the test environment, the tester will be able to control whether or not the internet connection goes down during the order and for how long the connection is down.
- Is the problem in any way specific to the Racinax 917X model and/or the colour? Isolation of the failure is needed, and then the report can be updated.
- Is the problem in any way specific to power surges? We would guess not but checking with different types of connections (during the isolation and replication discussed above) would make sense. If it proves independent of the type of connection, we would know that the problem is more general than just for power surges.
- The defect report should also include information about the date and time of the test, the beta test environment and the name of the beta tester and the tester entering the defect.

CHAPTER SIX

Tool support for testing

You may be wishing that you had a magic tool that would automate all of the testing for you. If so, you will be disappointed. However, there are a number of very useful tools that can bring significant benefits. In this chapter, we will see that there is tool support for many different aspects of software testing. We will see that success with tools is not guaranteed, even if an appropriate tool is acquired – there are also risks in using tools. There are some special considerations mentioned in the Syllabus for test execution tools and test management tools.

6.1 TEST TOOL CONSIDERATIONS

SYLLABUS LEARNING OBJECTIVES FOR 6.1 TEST TOOL CONSIDERATIONS (K2)

FL-6.1.1 Classify test tools according to their purpose and the test activities they support (K2)

FL-6.1.2 Identify benefits and risks of test automation (K1)

FL-6.1.3 Remember special considerations for test execution and test management tools (K1)

As we go through this section, watch for the Syllabus terms **data-driven testing**, **keyword-driven testing**, **performance testing tool**, **test automation**, **test execution tool** and **test management tool**. You will find these terms and their definitions, taken from the ISTQB Glossary, in this chapter, in the Glossary and online.

In this section, we will describe the various tool types in terms of their general functionality, rather than going into lots of detail. The reason for this is that, in general, the types of tool will be fairly stable over a longer period, even though there will be new vendors in the market, new and improved tools, and even new types of tool in the coming years.

We will also discuss both the benefits and the risks of using tools to run or execute tests. We will outline special considerations for test execution tools and for test management tools, the most popular types of test tools.

We will not mention any open source or commercial tools in this chapter. If we did, this book would date very quickly! Tool vendors are acquired by other vendors, change their names, and change the names of the tools quite frequently, so we will not mention the names of any tools or vendors.

There are a number of ways in which tools can be used to support testing activities:

- To start with the most visible use, we can use tools directly in testing. This includes test execution tools and test data preparation tools.
- We can use tools to help us manage the testing process. This includes tools that manage requirements, test cases, test procedures, automate test scripts, test results, test data and defects, as well as tools that assist with reporting and monitoring test execution progress.
- We can use tools as part of exploration, investigation and evaluation. For example, we can use tools to monitor file activity for an application.
- We can use tools in a number of other ways, in the form of any tool that aids in testing. This would include spreadsheets when used to manage test assets or progress, or as a way to document manual or automated tests.

6.1.1 Test tool classification

In this section, we will look first at how tools can be classified, starting with tool objectives or purposes. Then we will look at tools that support various aspects of testing: management of testing and testware, static testing, test design and implementation, test execution and logging, performance measurement and dynamic analysis, and finally tool support for specialized testing needs.

Tool classification

What are we trying to achieve with tools? What are the motivations? As you can imagine, these vary depending on the activity, but common objectives or purposes for the use of tools include the following:

- We might want to improve the efficiency of our test activities by automating repetitive tasks or by automating activities that would otherwise require significant resources to do manually. For example, the execution of regression tests is repetitive and takes significant time and effort to do manually.
- We might want to improve the efficiency of our testing by providing support for manual test activities throughout the test process. For example, tools can help in noting our findings while doing exploratory testing. See Chapter 1, Section 1.1.4 for more on the test process.
- We might want to improve the quality of test activities by achieving more consistency and reproducibility. For example, tools can help to reproduce failures more accurately and with more information. When a defect is fixed, the exact same test should be run again, otherwise you cannot be sure that the defect was correctly fixed. This requires both consistency and reproducibility.
- We might need to carry out activities that simply cannot be done manually, but which can be done via automated tools. For example, large-scale performance testing can only be achieved using tools; it is highly unfeasible to get thousands of users to test your system manually.
- We might want to increase the reliability of our testing. For example, automation of large data comparisons, simulating user behaviour, or repeating large numbers of tests (which can lead to mistakes due to tedium of the work, if it is done by people).

How are tools classified?

Tools can be classified in different ways. One is by their purpose (as above), or they can be classified by licencing model (commercial or open source), by price and/or by technology used. In this chapter, tools are classified by the testing activities or areas that are supported by a set of tools, for example, tools that support management activities, tools to support static testing, etc.

There is not necessarily a one-to-one relationship between a type of tool described here and a tool offered by a commercial tool vendor or an open source tool. Some tools perform a very specific and limited function and support only one test activity (sometimes called a point solution), but many of the tools provide support for a number of different functions (tool suites or families of tools). For example, a test management tool may provide support for managing testing (progress monitoring), configuration management of testware, defect management, and requirements management and traceability; another tool may provide both coverage measurement and test design support.

Tools versus people

There are some things that people can do much better or easier than a computer can do. For example, when you see a friend in an unexpected place, say in an airport, you can immediately recognize their face. This is an example of pattern recognition that people are very good at, but it is not easy to write software that can recognize a face.

There are other things that computers can do much better or more quickly than people can do. For example, can you add up 20 three-digit numbers quickly? This is not easy for most people to do, so you are likely to make some mistakes even if the numbers are written down. A computer does this accurately and very quickly. As another example, if people are asked to do exactly the same task over and over, they soon get bored and then start making mistakes.

The point is that it is a good idea to use computers to do things that computers are really good at and that people are not very good at. Tool support is very useful for repetitive tasks – the computer doesn't get bored and will be able to exactly repeat what was done before. Because the tool will be fast, this can make those activities much more efficient and more reliable. The tools can also do things that might overload a person, such as comparing the contents of a large data file or simulating how the system would behave.

Tools that affect their own results

A tool that measures some aspect of software may be intrusive. This means that the tool may have unexpected side effects in the software under test. For example, a tool that measures timings for performance testing needs to interact very closely with that software in order to measure it. A performance tool will set a start time and a stop time for a given transaction in order to measure the response time, for example. But the act of taking that measurement, that is, storing the time at those two points, could actually make the whole transaction take slightly longer than it would do if the tool was not measuring the response time. Of course, the extra time is very small, but it is still there. This effect is called the 'probe effect'.

Another example of the probe effect occurs with coverage tools. In order to measure coverage, the tool must first identify all of the structural elements that might be exercised to see whether a test exercises it or not. This is called instrumenting the code. The tests are then run through the instrumented code so that the tool can tell (through the instrumentation) whether or not a given

branch (for example) has been exercised. But the instrumented code is not the same as the real code – it also includes the instrumentation code. In theory, the code is the same, but in practice, it isn't. Because different coverage tools work in slightly different ways, you may get a slightly different coverage measure on the same program because of the probe effect. For example, different tools may count branches in different ways, so the percentage of coverage would be compared to a different total number of branches. The response time of the instrumented code may also be significantly worse than the code without instrumentation. (There are also non-intrusive coverage tools that observe the blocks of memory containing the object code to get a rough measurement without instrumentation, for example for embedded software.)

One further example of the probe effect is when a debugging tool is used to try to find a particular defect. If the code is run with the debugger, then the bug disappears; it only reappears when the debugger is turned off, thereby making it much more difficult to find. These are sometimes known as "Heisen-bugs" (after Heisenberg's uncertainty principle).

Tool classification categories

In the descriptions of the tools below, we will indicate the tools that are more likely to be used by developers, for example during component testing and component integration testing, by showing (D) (for Developer) after the tool type. For example, coverage measurement tools are most often used in component testing, but performance testing tools are more often used at system testing, system integration testing and acceptance testing.

Note that for the Foundation Certificate exam, you only need to recognize the different types of tools; you do not need to understand in any detail what they do (or know how to use them). We will briefly describe what the tools do (more detail than in the Syllabus) to help you identify the different types of tool.

Tool support for management of testing and testware

Test management tool A tool that provides support to the test management and control part of a test process. It often has several capabilities, such as testware management, scheduling of tests, the logging of results, progress tracking, incident management and test reporting.

(Note that the Glossary definition mentions incident management, but the Syllabus covers defect management.)

What does test management mean? It could be the management of tests or it could be managing the testing process. The tools in this broad category provide support for either or both of these. The management of testing applies over the whole of the software development life cycle, so a test management tool could be among the first to be used in a project. A test management tool may also manage the tests, which would begin early in the project and would then continue to be used throughout the project and also after the system had been released. In practice, test management tools are typically used by specialist testers or test managers at system or acceptance test level.

Test management tools and application lifecycle management (ALM) tools

Test management tools provide features that cover both the management of testing, such as progress reports and keeping track of testing activities, and the management of testware, such as logging of test results and keeping track of test environments needed for tests. There are some tools that provide support for only one activity (for example defect management tools); other tools or tool suites may provide support for all test management activities. Special considerations for test management tools are discussed in Section 6.1.3.

Application lifecycle management (ALM) tools manage not only testing but also development and deployment. With a focus on communication, collaboration and task tracking, they are popular in Agile development.

The information from test management tools (and ALM tools) can be used to monitor the testing process and decide what actions to take (test control), as described in Chapter 5. The tool also gives information about the component or system being tested (the test object). Test management tools help to gather, organize and communicate information about the testing on a project.

Requirements management tools

Are requirements management tools really testing tools? Some people may say they are not, but they do provide some features that are very helpful to testing. Because tests are based on requirements, the better the quality of the requirements, the easier it will be to write tests from them. It is also important to be able to trace tests to requirements and requirements to tests, as we saw in Chapter 2. Note that requirements means any source of what the system should do, such as user stories.

Defect management tools

This type of tool is also known as a defect tracking tool, an incident management tool, a bug tracking tool or a bug management tool. However not all of the things tracked are actually defects or bugs. There may also be perceived problems, anomalies (that aren't necessarily defects) or enhancement requests; this is why they are sometimes referred to as incidents and incident management tools. Also, what is normally recorded is information about the failure (not the defect) that was generated during testing; information about the defect that caused that failure would come to light when someone (for example a developer) begins to investigate the failure.

Defect reports go through a number of stages, from initial identification and recording of the details, through analysis, classification, assignment for fixing, fixed, re-tested and closed, as described in Chapter 5. Defect management tools make it much easier to keep track of the defects and their status over time.

Configuration management tools

An example: A test group began testing the software, expecting to find the usual fairly high number of problems. But to their surprise, the software seemed to be much better than usual this time. Very few defects were found. Before they celebrated the great quality of this release, they just made an additional check to see if they had the right version and discovered that they were actually testing the version from two months earlier (which had been debugged) with the tests for that earlier version. It was nice to know that this was still OK, but they were not actually testing what they thought they were testing or what they should have been testing.

Configuration management tools are not strictly testing tools either, but good configuration management is critical for controlled testing, as was described in Chapter 5. We need to know exactly what it is that we are supposed to test, such as the exact version of all of the things that belong in a system. It is possible to perform configuration management activities without the use of tools, but the tools make life a lot easier, especially in complex environments. The same tool may be used for testware as well as for software items. Testware also has different versions and is changed over time. It is important to run the correct version of the tests as well, as our example shows.

Continuous integration tools (D)

The (D) after this (and other types of tools) indicates that these tools are more likely to be used by developers.

Incremental and iterative development life cycles require frequent builds, and continuous integration tools are an essential part of the Agile toolkit. Continuous integration is the practice of integrating new or changed code with the existing code repository very frequently, at least daily but sometimes dozens of times a day or more. Unit tests are most often automatically run when a new build is made (when a developer commits his or her change), so any defects found can be corrected immediately. The result of the integration is a system that is tested and could in principle be deployed to users at any time.

If the deployment is automated, then it is called continuous delivery, but there might still be some final human approval before it is released. If deployment is automatic (with no final human approval), then it is called continuous deployment. Continuous integration, delivery and deployment are the basis for DevOps, where code changes are delivered to users in an automated delivery pipeline.

These tools are continuous because the integration of the code components happens so often, and is automatic (without human intervention), whenever the new build is triggered by a developer. This type of tool is included here because it includes tests, which are managed in the continuous integration tool.

Tool support for static testing

The tools described in this section support the testing activities described in Chapter 3.

Tools that support reviews

The value of different types of review was discussed in Chapter 3. For a very informal review, where one person looks at another's work product and gives a few comments about it, a tool such as this might just get in the way. However, when the review process is more formal, when many people are involved, or when the people involved are in different geographical locations, then tool support becomes far more beneficial.

It is possible to keep track of all the information for a review process using spreadsheets and text documents, but a review tool that is designed for the purpose is more likely to do a better job. For example, one thing that should be monitored for each review is that the reviewers have not gone over the work product too quickly, that is, that the checking rate (number of pages checked per hour) was close to that recommended for that review cycle. A review tool could automatically calculate the checking rate and flag exceptions. The review tools can normally be tailored for the particular review process or type of review being done.

Static analysis tools (D)

Static analysis tools are normally used by developers as part of the development and component testing process. The key aspect is that the code (or other artefact) is not executed or run. Of course, the tool itself is executed, but the source code we are interested in is the input data to the tool.

Static analysis tools are an extension of compiler technology, in fact some compilers do offer static analysis features. It's worth checking what is available from existing compilers or development environments before looking at acquiring a more sophisticated static analysis tool.

Static analysis can also be carried out on things other than software code, for example, static analysis of requirements or static analysis of websites to assess for proper use of accessibility tags or the following of HTML standards.

Static analysis tools for code can help the developers to understand the structure of the code and can also be used to enforce coding standards.

Tool support for test design and specification

The tools described in this section support the testing activities described in Chapter 4. These tools help to create maintainable test designs, test cases, test procedures and test data.

Test design tools

Test design tools help to construct test cases, or at least test inputs (part of a test case). If an automated oracle is available, then the tool can also construct the expected result, so it can actually generate test cases, rather than just test inputs.

Tests that are designed using a tool need to have a starting point to derive the tests from. This could be requirements from a requirements management tool, which may identify valid and invalid boundary values for an input field, for example. Tests can also be derived from elements on a screen, to ensure that each element is exercised by a test, for example buttons, input fields or pull-down lists. Tests can also be derived from a model of the system, as in model-based testing (see below). Coverage tools may identify the tests needed to extend white-box coverage.

There are two problems with test design tools. First, although it is relatively easy to generate test inputs, automatically generating the correct expected result is more challenging. Sometimes an oracle is available (for example valid boundary conditions), other times only a partial oracle (generating some aspect but not all detail of an expected result), and most often no oracle other than 'The system is still running'.

The second problem is having too many tests and needing to find a way of identifying the most important tests to run. Cutting down an unmanageable number of tests can be done by risk analysis (see Chapter 5).

Model-based testing tools

Tools that generate test inputs or test cases from stored information that represents some kind of model of the system or software are called model-based testing tools. Such tools may be based on a state diagram, to implement state transition testing, for example.

Model-based testing tools work by generating tests (inputs or test cases) automatically based on what is stored about the model used to describe the system behaviour. If the system is changed, only the model is updated, and the new tests are then generated automatically.

More information about Model-Based Testing (MBT) is available in the ISTQB Foundation Level Model-Based Tester Extension Syllabus and qualification.

Test data preparation tools

Setting up test data can be a significant effort, especially if an extensive range or volume of data is needed for testing. Test data preparation tools help in this area. They may be used by developers, but they may also be used during system or acceptance testing. They are particularly useful for performance and reliability testing, where a large amount of realistic data is needed.

The most sophisticated tools can deal with a range of files and database formats. These tools are also useful for anonymizing data to conform to data protection rules.

Test-driven development (TDD) tools (D)

Test-driven development (TDD) is a software development approach started as part of EXtreme Programming, which had a great influence on Agile development. The idea behind TDD is that tests are written first, before code, because if you think about

how you will test something, you are anticipating what could go wrong; when you then write the code, it is more likely to cope with problems.

TDD tools provide a framework for writing and running tests. The tests that are developed in TDD are basically unit tests; these are not sufficient testing for the component being developed, but they are a good starting point. Other levels of testing are still needed, that is, integration, system and acceptance testing.

Acceptance test-driven development (ATDD) and behaviour-driven development (BDD) tools

Tools to develop system-level tests before developing the system itself are becoming increasingly popular. ATDD is a way of capturing requirements by writing acceptance tests in conjunction with users. This approach is also called Specification by Example, as in Adzic [2011]. BDD is focused on system behaviour and functionality, and is normally done by the development team. Tools to support ATDD /BDD usually have syntax (rules), like a natural language, that need to be followed. An example of this syntax is the 'Given/When/Then' format: 'Given <some condition>, When <something is done>, Then <some result should happen>'. Another format is 'As a <role, for example customer>, In order to <achieve something, for example buy a product>, I want to <do something, for example place an order from my phone>'.

The ATDD/BDD tools support this domain language for specifying the tests, and also enable those tests to be generated and run.

Tools may cover both test design/implementation and test execution/logging. Both TDD and ATDD/BDD tools enable the tests specified in the tools to be run, so they cover test execution as well as test design and implementation. In addition, some other test design and integration tools, for example MBT tools, provide outputs directly to test execution and logging tools. These tools were described in this section, but also provide support for the test activities described below.

Tool support for test execution and logging

Test execution tools

When people think of a testing tool, it is usually a test execution tool that they have in mind, a tool that can run tests. This type of tool is also referred to as a test running tool. Some tools of this type offer a way to get started by capturing or recording manual tests; hence they are also known as capture/playback tools, capture/replay tools or record/playback tools. The analogy is with recording a television programme and playing it back. However, the tests are not something which is played back just for someone to watch!

Test execution tool
A test tool that executes tests against a designated test item and evaluates the outcomes against expected results and postconditions.

Test execution tools use a scripting language to drive the tool. The scripting language is actually a programming language. Testers who wish to use a test execution tool directly may need to use programming skills to create and modify the scripts, although some newer tools provide a higher-level interface for testers. The advantage of programmable scripting is that tests can repeat actions (in loops) for different data values (test inputs), they can take different routes depending on the outcome of a test (for example if a test fails, go to a different set of tests) and they can be called from other scripts giving modular structure to the set of tests.

When people first encounter a test execution tool, they tend to use it to capture (or record) tests, which sounds really good when you first hear about it. However, the approach breaks down when you try to replay the captured tests. This approach does

not scale up for large numbers of tests, as described in Section 6.1.3. The main reason for this is that a captured script is very difficult to maintain because:

- It is closely tied to the flow and interface presented by the GUI (graphical user interface).
- It may rely on the circumstances, state and context of the system at the time the script was recorded. For example, a script will capture a new order number assigned by the system when a test is recorded. When that test is played back, the system will assign a different order number and reject subsequent requests that contain the previously captured order number.
- The test input information is hard-coded, that is, it is embedded in the individual script for each test.

Any of these things can be overcome by modifying the scripts, but then we are no longer just recording and playing back! If it takes more time to update a captured test than it would take to run the same test again manually, the scripts tend to be abandoned and the tool becomes 'shelf-ware'.

There are better ways to use test execution tools to make them work well and actually deliver the benefits of unattended automated test running. There are at least five levels of scripting and also different comparison techniques. Data-driven testing is an advance over captured scripts, but keyword-driven testing gives significantly more benefits. See Fewster and Graham [1999], Buwalda *et al.* [2001], Graham and Fewster [2012], Gamba and Graham [2018] and Section 6.2.3.

There are many different ways to use a test execution tool and the tools themselves are continuing to gain new useful features. Test execution tools are often used for regression testing, where tests are run that have been run before, such as in continuous integration. One of the most significant benefits of using this type of tool is that whenever an existing system is changed (for example for a defect fix or an enhancement), the tests that were run earlier can be run again, to make sure that the changes have not disturbed the existing system.

Coverage tools (for example requirements coverage, code coverage (D))

How thoroughly have you tested? Coverage tools can help answer this question. Requirements coverage tools measure how many requirements have been exercised by a set of tests; code coverage tools (D), normally used by developers, measure how many code elements, such as statements or branches, have been exercised by a set of tests.

A coverage tool first identifies the elements or coverage items that can be counted, and where the tool can identify when a test has exercised that coverage item. At component testing level, the coverage items could be lines of code or code statements or decision outcomes (for example the True or False exit from an IF statement). At component integration level, the coverage item may be a call to a function or module. At system or acceptance level, the coverage item may be a user story, a requirement, a function or a feature.

The coverage tool counts the number of coverage items that have been executed by the test suite and reports the percentage of coverage items that have been exercised and may also identify the items that have not yet been exercised (that is, not yet tested). Additional tests can then be run to increase coverage.

Note that the coverage tools only measure the coverage of the items that they can identify. Just because your tests have achieved 100% coverage, this does not mean that your software is 100% tested! See also Chapter 4 Section 4.3.

Test harnesses (D)

Some software components cannot be tested independently, for example when they depend on receiving data from other components, or they need to send a message to a different component or system and receive a reply. Waiting until such components are integrated with the whole system is asking for trouble. It is much better to test on as small a scale as possible. In this and similar cases, the component can be tested if the data it needs can be supplied to it in another way. A driver is another component or tool that will call or invoke the component we want to test and supply it with the data it needs. A stub is a small component that interacts with the tested component in the way it expects, but just for that purpose, so it pretends to be the part that receives the message and returns a reply. Stubs may also be referred to as mock objects.

A test harness is the test environment that provides these drivers and stubs so that our component can be tested as much as possible on its own.

Unit test framework tools (D)

A unit test framework is another tool which is used by developers when testing individual components (or sets of integrated components). Unit test frameworks can be used in Agile development to automate tests in parallel with development. Both unit test frameworks and test harnesses enable the developer to test, identify and localize any defects. The framework can also act as a driver and/or as a mock object to supply any information needed by the software being tested (for example an input that would have come from a user) and also receive any information sent by the software (for example a value to be displayed on a screen).

Test harnesses or unit test frameworks may be developed in-house for particular systems.

There are a large number of xUnit tools for different programming languages, for example JUnit for Java, NUnit for .Net applications, etc. There are both commercial tools and also open source tools. Unit test framework tools are very similar to test execution tools, since they include facilities such as the ability to store test cases and monitor whether tests pass or fail, for example. The main difference is that there is no capture/playback facility and they tend to be used at a lower level, that is, for component or component integration testing, rather than for system or acceptance testing. Unit test frameworks may also provide support for measuring coverage.

Tool support for performance measurement and dynamic analysis

The tools described in this section support testing that can be carried out on a system when it is operational, that is, while it is running. This can be during testing or could be after a system is released into live operation. These tools support activities that cannot reasonably be done manually.

Performance testing tools

Performance testing tool A test tool that generates load for a designated test item and that measures and records its performance during test execution.

Performance testing tools are concerned with testing at system level to see whether or not the system will stand up to a high volume of usage. A load test checks that the system can cope with its expected number of transactions. A volume test checks that the system can cope with a large amount of data, for example many fields in a record, many records in a file, etc. A stress test is one that goes beyond the normal expected usage of the system (to see what would happen outside its design expectations), with respect to load or volume.

In performance testing, many test inputs may be sent to the software or system where the individual results may not be checked in detail. The purpose of the test is to measure characteristics, such as response times, throughput or the mean time between failures (for reliability testing). Load generation can simulate multiple users or high volumes of input data. More sophisticated tools can generate different user profiles, different types of activity, timing delays and other parameters. Adequately replicating the end-user environments or user profiles is usually key to realistic results. Performance testing tools normally provide reports based on test logs and graphs of load against response times.

Analyzing the output of a performance testing tool is not always straightforward and it requires time and expertise. If the performance is not up to the standard expected, then some analysis needs to be performed to see where the problem is and to know what can be done to improve the performance.

Monitoring tools

Monitoring tools are used to continuously keep track of the status of the system in use, in order to have the earliest warning of problems and to improve service. There are monitoring tools for servers, networks, databases, security, performance, website and internet usage, and applications.

Monitoring tools may help to identify performance problems and network problems, and give information about the use of the system, such as the number of users.

Dynamic analysis tools (D)

Dynamic analysis tools are so named because they require the code to be running. They are analysis rather than testing tools because they analyze what is happening behind the scenes while the software is running (whether being executed with test cases or being used in operation).

An analogy with a car may be useful here. If you go to look at a car to buy, you might sit in it to see if is comfortable and see what sound the doors make. This would be static analysis because the car is not being driven. If you take a test drive, then you would check that the car performs as you expect (for example, it turns right when you turn the steering wheel clockwise). This would be a test. While the car is running, if you were to check the oil pressure or the brake fluid, this would be dynamic analysis. It can only be done while the engine is running, but it is not a test case.

When your PC's response time gets slower and slower over time, but is much improved after you re-boot it, this may well be due to a memory leak, where the programs do not correctly release blocks of memory back to the operating system. Eventually the system will run out of memory completely and stop. Rebooting restores all of the memory that was lost, so the performance of the system is now restored to its normal state.

Dynamic analysis tools would typically be used by developers in component testing and component integration testing.

Tool support for specialized testing needs

In addition to the tools that support specific testing activities, there are tools that support other testing needs and more specific testing issues. Tool support is available to support these other testing needs.

Data quality assessment

Data quality is very important, and there are tools to assess it. In many IT-centric organizations, systems and 'systems of systems' manage very large volumes of complex, interrelated data. This puts data at the centre of many projects in these

organizations. These tools can check data according to given validation rules, for example, checking that a particular field is numeric or of a given length. Any data that fails a check is reported for someone to look at and fix.

Data conversion and migration

When technology moves on, it often becomes necessary to have to transfer large sets of data from one type of storage to another. The data involved can vary in terms of criticality and volume. Data conversion and migration tools can review and verify data conversion and check that data has been migrated according to the migration rules. For example, a field in the new database may be larger than the equivalent field in the old one, so the value stored may need to have extra characters or digits added to it. These tools can help ensure that the processed data is correct and complete, and that it complies to pre-defined, context-specific standards, even when the volume of data is large.

Usability testing

Tools to support usability testing can help to assess user experience in using the system, for example, surveys after using a website or mobile app. Tools can check to make sure there are no broken links on a web page. Tools can also monitor usage, such as which links are clicked most often. Video recorders and screen capture utilities can also be used to help assess usability. Major companies that sell to the public may also have a usability lab, where beta versions of software are used by volunteers from the target market; this could include video, analysis of key presses or even eye movement tracking.

Accessibility testing

Regulations now require that systems including websites are usable by people with various disabilities. For example, it should be possible to increase the font size of text. For blind users, each visual object on a screen should have alternative text associated with it which can be read by a screen reader. For example, a field with a magnifying glass on the left would have the alternative text 'Search'; an image of a horse galloping would have the text 'Galloping horse'. Alternative text is particularly important for buttons and interactive images, such as the 'Submit' button. See www.w3.org for more information about accessibility for websites.

Localization testing

This is also called internationalization. Localization is making sure that systems are understood in different countries and in different cultures. This involves translation of text for information, menu items, buttons, error messages, field names: in fact anything that someone will read on a page. Although tools can help in some ways with translations, the tools often get it wrong, particularly with colloquial expressions. Tools can be of more help in other ways, for example if the same translation is used in many different environments or platforms (such as mobile devices). This is one area where human intelligence is still needed.

Security testing

There are a number of tools that protect systems from external attack, for example firewalls, which are important for any system.

Security testing tools can be used to test security by trying to break into a system, whether or not it is protected by a security tool. The attacks may focus on the network, the support software, the application code or the underlying database.

Security testing tools can provide support in identifying viruses, detecting intrusions such as denial of service attacks, simulating various types of external attacks, probing for open ports or other externally visible points of attack, and identifying weaknesses in password files and passwords. In addition, these tools can perform security checks during operation, for example checking the integrity of files, intrusion detection and the results of test attacks.

Portability testing

Portability testing tools help to support testing on different platforms and in different environments. You may have an application that needs to perform the same functions on desktop computers, laptops, tablets and mobile phones. Although the functionality is the same, the way in which the information is displayed would be different. Portability testing tools can help to run the same tests on the different devices, even though the look and feel of each interface is different.

About tools in general

In this chapter, we have described tools according to their general functional classifications. There are also further specializations of tools within these classifications. For example, there are web-based performance testing tools as well as performance testing tools for back-office systems. There are static analysis tools for specific development platforms and programming languages, since each programming language and every platform has distinct characteristics. There are dynamic analysis tools that focus on security issues, as well as dynamic analysis tools for embedded systems.

Tool sets may be bundled for specific application areas such as web-based, embedded systems or mobile applications.

6.1.2 Benefits and risks of test automation

The reason for acquiring tools to support testing is to gain benefits, by using software to do certain tasks that are better done by a computer than by a person.

The term **test automation** is most often used to refer to tools that execute tests and compare results, that is, test execution tools. This type of tool is one which can provide great benefits, but there are also significant risks. Test automation is also much broader than just supporting execution; tools are available to support many aspects of testing.

Test automation The use of software to perform or support test activities, for example, test management, test design, test execution and results checking.

Potential benefits of using tools

There are many benefits that can be gained by using tools to support testing, whatever the specific type of tool. Benefits include:

Reduction in repetitive manual work

Repetitive work is tedious to do manually. People become bored and make mistakes when doing the same task over and over. Examples of this type of repetitive work include running regression tests, entering the same test data over and over again (both of which can be done by a test execution tool), checking against coding standards (which can be done by a static analysis tool) or running a large number of tests through the system in a short time (which can be done by a performance testing tool). Tools can also help to set up or tear down test environments. Using tools to help with repetitive work can save time (as well as frustration).

Greater consistency and repeatability

People tend to do the same task in a slightly different way even when they think they are repeating something exactly. A tool will exactly reproduce what it did before, so each time it is run, the result is consistent. Examples of where this aspect is beneficial include checking to confirm the correctness of a fix to a defect (which can be done by a debugging tool or test execution tool), entering test inputs (which can be done by a test execution tool) and generating tests from requirements (which can be done by a test design tool, MBT tool or possibly a requirements management tool).

More objective assessment

If a person calculates a value from the software or defect reports, they may inadvertently omit something, or their own subjective prejudices may lead them to interpret that data incorrectly. Using a tool means that subjective bias is removed and the assessment is more repeatable and consistently calculated. Examples include assessing the structure of a component (which can be done by a static analysis tool), measuring coverage of the test item by a set of tests (coverage measurement tool), assessing system behaviour (monitoring tools) and defect statistics (test management tool or defect management tool).

Easier access to information about testing

Having lots of data does not mean that information is communicated. Information presented visually is much easier for the human mind to take in and interpret. For example, a chart or graph is a better way to show information than a long list of numbers. This is why charts and graphs in spreadsheets are so useful. Special-purpose tools give these features directly for the information they process. Examples include statistics and graphs about test progress (test execution tool or test management tool), defect rates (defect management or test management tool) and performance (performance testing tool).

In addition to these general benefits, each type of tool has specific benefits relating to the aspect of testing that the particular tool supports. These benefits are normally prominently featured in the information available for the type of tool. It is worth investigating a number of different tools to get a general view of the benefits.

Risks of using tools

Although there are significant benefits that can be achieved using tools to support testing activities, there are many organizations that have not achieved the benefits they expected.

Simply purchasing a tool is no guarantee of achieving benefits, just as buying membership in a gym does not guarantee that you will be fitter. Each type of tool requires investment of effort and time in order to achieve the potential benefits.

There are many risks that are present when tool support for testing is introduced and used, whatever the specific type of tool. Risks include:

Expectations for the tool may be unrealistic (including functionality and ease of use)

Unrealistic expectations may be one of the greatest risks to success with tools. The tools are only software and we all know that there are many problems with any kind of software! It is important to have clear objectives for what the tool can do and that those objectives are realistic. Unrealistic expectations may be both for functionality and for ease of use.

The time, cost and effort for the initial introduction of a tool may be under-estimated (including training and external expertise)

Introducing something new into an organization is seldom straightforward. Having purchased a tool, you will want to move from downloading the tool to having a number of people being able to use the tool in a way that will bring benefits. There will be technical problems to overcome, but there will also be resistance from other people – both need to be addressed in order to succeed in introducing a tool. Often forgotten are the time, cost and effort or external expertise to help get things onto the right track, and for training, not just in the use of the tool itself, but internal training after a pilot project so that everyone is using the tool and agreed internal conventions in a consistent way for maximum benefit. Simply acquiring a tool and getting it started is not enough either. When a tool is introduced, it will affect the way testing is done, so test processes and test documentation will need to change.

The time and effort needed to achieve significant and continuing benefits from the tool may be under-estimated

Think back to the last time you did something new for the very first time (learning to drive, riding a bike, skiing). Your first attempts were unlikely to be very good, but with more experience you became much better. Using a testing tool for the first time will not be your best use of the tool either. It takes time to develop ways of using the tool in order to achieve what's possible. In the excitement of acquiring a new tool, it is easy to forget about things like the need for changes in the test process as the tool is implemented. These changes should be planned from the start.

But it doesn't stop there. It is critical for continuing success that there is continuous improvement in the way the tool is used. This may involve re-factoring automation assets from time to time or changing the way those assets are organized. Fortunately, there are some short cuts (for example reading books and articles about other people's experiences and learning from them). See also Section 6.2.1 for more detail on introducing a tool into an organization.

The effort required to maintain the test assets generated by the tool may be under-estimated

Insufficient planning for maintenance of the assets that the tool produces and uses is a strong contributor to tools that end up as shelf-ware, along with the previously listed risks. Although particularly relevant for test execution tools, planning for maintenance is also a factor with other types of tool.

The tool may be relied on too much (seen as a replacement for test design or execution, or the use of automated testing where manual testing would be better)

Tools are definitely not magic! They can do very well what they have been designed to do (at least a good quality tool can), but they cannot do everything. A tool can certainly help, but it does not replace the intelligence needed to know how best to use it, and how to evaluate current and future uses of the tool. A test execution tool does not replace the need for good test design and should not be used for every test. Some tests are still better executed manually. For example, a test that takes a very long time to automate and will not be run very often is better done manually.

Version control of test assets or interoperability may be neglected

Configuration management is important for test assets, and that definitely includes automation assets. If this is neglected, it can lead to significant and unnecessary problems.

Relationships and interoperability issues between critical tools may be neglected, such as requirements management tools, configuration management tools, defect management tools, and tools from multiple vendors. Most organizations have a number of different tools, often to support different aspects of development and testing. Some organizations may have a tool suite with many different tool types bundled together, others may have a number of tools that are sourced from a variety of sources. The relationships and interoperability between the various tools can cause problems. These should be taken into account when acquiring a new tool, but when tools change, this may also cause interoperability problems to change. This is particularly important for continuous integration tools, but the problems there will become obvious quickly. Problems with integrating other tools may not be so obvious, but may store up problems for later.

The tool vendor may go out of business, retire the tool, or sell the tool to a different vendor

A commercial tool vendor may not be able to continue to support a tool that you have come to rely on. If a tool is acquired by a different organization, they may not have the same priorities. Sometimes tools that have been very popular are phased out. Open source tools also change over time and may become less suitable for you. When you choose a tool, always think about how you would cope if you had to change to a different tool, and structure your automation to cope with this, even if it seems very unlikely now.

The vendor may provide a poor response for support, upgrades and defect fixes

Commercial vendors, or those supporting open source tools, may not respond quickly or adequately to questions, problems or bugs found in the tools (yes it does happen!). Upgrades may cause problems for you because of the way that you have used the tool, but the vendor may not be interested in helping you if you are in a small minority of their customers.

An open source project may be suspended

Even open source projects can be affected. Since they are very much supported by a community, if the people involved are no longer active, it may be difficult to get help with tool problems.

A new platform or technology may not be supported by the tool

Your own applications are changing over time as well, moving onto new platforms and using new technology. If you are in the forefront of your industry, your testing tools may not yet be able to deal with these innovations.

There may be no clear ownership of the tool (for example for mentoring, updates etc.)

There should be one person who is responsible for technical aspects of the tool. This person may be an enthusiastic champion for automation in general, or they may have a more technical role and be the person who deals with licences, reporting problems, installing updates, etc. If no one takes ownership of the tool, there will be problems in the future, and the automation will begin to decay.

This list of risks is not exhaustive. Two other important factors are:

- The skill needed to create good tests.
- The skill needed to use the tools well, depending on the type of tool.

The skills of a tester are not the same as the skills of the tool user. The tester concentrates on what should be tested, what the test cases should be and how to prioritize the testing. The tool user concentrates on how best to get the tool to do its job effectively and how to give increasing benefit from tool use.

6.1.3 Special considerations for test execution and test management tools

Test execution tools

In order to know what tests to execute and how to run them, the test execution tool must have some way of knowing what to do – this is the script for the tool. But since the tool is only software, the script must be completely exact and unambiguous to the computer, which has no common sense. This means that the script is a program, written in a programming language. The scripting language may be specific to a particular tool, or it may be a more general language. Scripting languages are not used just by test execution tools, but the scripts used by the tool are stored electronically to be run when the tests are executed under the tool's control.

There are tools that can generate scripts by identifying what is on the screen rather than by capturing a manual test, but they still generate scripts to be used in execution: they are not 'script-free' or 'code-free'. This later generation of tools may use smart image-capturing technology and they can help to generate initial scripts quickly, so may be more useful than the traditional capture/replay tools. However, maintenance will still be needed on the scripts over time as the user interface evolves, for example. This will need someone to be able to work directly with the scripting language. The sales pitches for these tools may imply that this is not necessary, but this is misleading.

There are different levels of scripting. Five are described in Fewster and Graham [1999]:

- Linear scripts, which could be created manually or captured by recording a manual test.
- Structured scripts, using selection and iteration programming structures.
- Shared scripts, where a script can be called by other scripts so can be re-used: shared scripts also require a formal script library under configuration management.
- Data-driven scripts, where test data is in a file or spreadsheet to be read by a control script.
- Keyword-driven scripts, where all of the information about the test is stored in a file or spreadsheet, with a single control script that implements the tests described in the file using shared and keyword scripts.

Capturing a manual test seems like a good idea to start with, particularly if you are currently running tests manually anyway. But a captured test (a linear script) is not a good solution, for a number of reasons, including:

- The script does not know what the expected result is until you program it in. It only stores inputs that have been recorded, not test cases.
- A small change to the software may invalidate dozens or hundreds of scripts.
- The recorded script can only cope with exactly the same conditions as when it was recorded. Unexpected events (for example a file that already exists) will not be interpreted correctly by the tool.

However, there are some times when capturing test inputs (that is, recording a manual test) is useful. For example, if you are doing exploratory testing or if you are running unscripted tests with experienced business users, it can be very helpful simply to record everything that is done, as an audit trail. This serves as a form of documentation of what was tested (although analyzing it may not be easy). This audit trail can also be very useful if a failure occurs which cannot be easily reproduced. The recording of the specific failure can be played to the developer to see exactly what sequence caused the problem.

Captured test inputs can be useful in the short term, where the context remains valid. Just don't expect to replay them as regression tests (when the context of the test may be different). Captured tests may be acceptable for a few dozen tests, where the effort to update them when the software changes is not very large. Don't expect a linear scripting approach to scale to hundreds or thousands of tests.

Capturing tests does have a place, but it is not a large place in terms of automating test execution.

Data-driven testing A scripting technique that stores test input and expected results in a table or spreadsheet, so that a single control script can execute all of the tests in the table. Data-driven testing is often used to support the application of test execution tools such as capture/playback tools.

Keyword-driven testing (action word-driven testing) A scripting technique that uses data files to contain not only test data and expected results, but also keywords related to the application being tested. The keywords are interpreted by special supporting scripts that are called by the control script for the test.

Data-driven testing uses scripts that allow the data, that is, the test inputs and expected outcomes, to be stored separately from the script. This can include situations where, instead of the tester putting hard-coded data combinations into a spreadsheet, a tool generates and supplies data in real time, based on configurable parameters. For example, a tool may generate a random user ID for accounts and can take advantage of random number seeding to ensuring repeatability in the pattern of user IDs. Whichever way data-driven testing is implemented, it has the advantage that a tester who does not know how to use a scripting language can populate a file or spreadsheet with the data for a specific test. This is particularly useful when there are a large number of data values that need to be tested using the same control script.

Keyword-driven testing uses scripts that include not just data but also keywords in the data file or spreadsheet. In other words, the spreadsheet or data file contains keywords that describe the actions to be taken and the test data. Action words is another term used for keywords as described in Buwalda *et al.* [2001]. Keyword-driven (or action word) automation enables a tester (who is not a script programmer) to devise a great variety of tests, not just varied input data for essentially the same test, as in data-driven scripts. The tester needs to know what keywords are currently available to use (by someone having written a script for it) and what data the keyword is expecting, but the tester can then write tests, not just enter specific test data. The tester can also request additional keywords to be added to the available programmed set of scripts as needed. Keywords can deal with both test inputs and expected outcomes.

Someone still needs to be able to use the tool directly and be able to program in the tool's scripting language, in order to write and debug the keyword scripts that will use the data tables or keyword tables. A small number of automation specialists can support a larger number of testers, who then do not need to learn to be script programmers (unless they want to).

The data files (data-driven or keyword-driven) include the expected results for the tests. The actual results from each test run also need to be stored, at least until they are compared to the expected results and any differences are logged.

Whatever form of scripting is used, there are two important points to remember:

- Someone needs to have expertise in the scripting language of the tool.
- The expected results of a test still need to be compared to the actual result produced by the application when the (automated) test is run.

There are two types of result comparison: dynamic, which is done while the test is running, or post-execution, which is done after the test completes. Dynamic comparison is best for things like pop-ups or error messages during a test, and can help to determine the subsequent progress of the test. Post-execution comparison is best for comparing large volumes of data, for example in a database, after tests have made many changes.

Model-based testing tools include not only test execution capability, but can also generate test inputs and expected results. The tool stores a functional specification such as an activity diagram or finite state machine. This is generally specified by a system designer. The MBT tool uses the stored model to generate inputs and expected results. For example, a state diagram shows the type of input that would generate a state transition. The tool selects a sample value from the input space, and the expected result would include the end state after the transition. The generated test cases can be stored in a test management tool or run by a test execution tool. Note that there is an ISTQB qualification for MBT: Foundation Level Model-Based Testing (an extension to the Foundation level).

More information on data-driven and keyword-driven scripting can be found in Fewster and Graham [1999], Buwalda et al [2001], Janssen and Pinkster [2001], Graham and Fewster [2012] and Gamba and Graham [2018]. Note that there is an ISTQB qualification in test automation: Advanced Level Test Automation Engineer.

Test management tools

Test management tools can provide a lot of useful information, but the information as produced by the tool may not be in the form that will be most effective within your own context. Some additional work may be needed to produce interfaces to other tools or a spreadsheet in order to ensure that the information is communicated in the most effective way.

Test management tools need to interface with other tools (including spreadsheets for example) for various reasons, including:

- To produce useful information in a format that fits the needs of the organization.
- To maintain consistent traceability to requirements in a requirements management tool.
- To link with test object version information in the configuration management tool.

Tools such as application lifecycle management (ALM) tools may have aspects that are used by different people within the organization. For example, a high-level manager wants to see trends and graphs about test progress, application quality schedules and budgets, but a developer wants to see detailed information about how a defect occurred.

A report produced by a test management tool (either directly or indirectly through another tool or spreadsheet) may be a very useful report at the moment, but the same information may not be useful in three or six months. It is important to monitor the information produced to ensure it is the most relevant now.

It is important to have a defined test process before test management tools are introduced. If the testing process is working well manually, then a test management tool can help to support the process and make it more efficient. If you adopt a test management tool when your own testing processes are immature, one option is to follow the standards and processes that are assumed by the way the tool works.

This can be helpful, but it is not necessary to follow the processes specific to the tool. The best approach is to define your own processes, taking into account the tool you will be using, and then adapt the tool to provide the greatest benefit to your organization.

6.2 EFFECTIVE USE OF TOOLS

SYLLABUS LEARNING OBJECTIVES FOR 6.2 EFFECTIVE USE OF TOOLS (K1)

FL-6.2.1 Identify the main principles for selecting a tool (K1)

FL-6.2.2 Recall the objectives for using pilot projects to introduce tools (K1)

FL-6.2.3 Identify the success factors for evaluation, implementation, deployment, and on-going support of test tools in an organization (K1)

In this section, we discuss the principles and process of introducing tools into your organization. We look at the process of tool selection. We'll talk about how to carry out tool pilot projects. We'll conclude with thoughts on what makes tool introduction successful. There are no Glossary terms for this section.

Lots of tool advice is online, and good advice on automation can be found in Fewster and Graham [1999 and 2012], Gamba and Graham [2018] and Axelrod [2018].

6.2.1 Main principles for tool selection

The place to start when introducing a tool into an organization is not with the tool: it is with the organization. In order for a tool to provide benefit, it must match a need within the organization, and solve that need in a way that is both effective and efficient. The tool should help to build on the strengths of the organization and address its weaknesses. The organization needs to be ready for the changes that will come with the new tool. If the current testing practices are not good and the organization is not mature, then it is generally more cost-effective to improve testing practices rather than to try to find tools to support poor practices. Automating chaos just gives faster chaos!

Of course, we can sometimes improve our own processes in parallel with introducing a tool to support those practices. We can pick up some good ideas for improvement from the ways that the tools work. However, be aware that the tool should not take the lead, but should provide support to what your organization defines.

The following factors are important in selecting a tool:

- Assessment of the organization's maturity (for example strengths and weaknesses, readiness for change).
- Identification of the areas within the organization where tool support will help to improve testing processes.
- Understanding the technologies used by the test object(s), so that a tool will be selected that is compatible with those technologies.

- Knowledge of any build and continuous integration tools already being used within the organization, to make sure that the new tool(s) will integrate with them and be compatible.
- Evaluation of tools against clear requirements and objective criteria.
- Consideration of any free trial period for the tool (for commercial tools) to ensure that this gives adequate time to evaluate the tool.
- Evaluation of the vendor (including training, support and other commercial aspects) or support for non-commercial tools (open source).
- Identification of internal requirements for coaching and mentoring in the use of the tool.
- Evaluation of training needs for those who will use the tools directly and indirectly (for example without technical detail), taking into account testing skills and test automation skills (for those working directly with the tools).
- Consideration of pros and cons of different licencing models (for example commercial or open source).
- Estimation of a cost-benefit ratio based on a concrete and realistic business case (if required).

A final step would be to do a proof-of-concept evaluation. The purpose of this is to see whether or not the tool performs as expected, both with the software under test and with any other tools or aspects of your own infrastructure. For example, you want to find out now if your proposed tool cannot communicate with your configuration management system. You may also need to identify changes to your current infrastructure to enable the new tool(s) to be used effectively.

6.2.2 Pilot project

After selecting a tool (or tools) and a successful proof-of-concept, the next step is to have a pilot project as the first step in using the tool(s) for real. The pilot project will use the tool in earnest but on a small scale, with sufficient time to explore different ways of using the tool. Objectives should be set for the pilot in order to assess whether or not the tool can accomplish what is needed within the current organizational context.

A pilot tool project should expect to encounter problems. They should be solved in ways that can be used by everyone later on. The pilot project should experiment with different ways of using the tool. For example, different reports from a test management tool, different scripting and comparison techniques for a test execution tool or different load profiles for a performance testing tool.

The objectives for a pilot project for a newly acquired tool are:

- To gain in-depth knowledge about the tool (more detail, more ways of using it) and to understand more fully both its strengths and its weaknesses.
- To see how the tool would fit with existing processes and practices, determining how those would need to adapt and change to work well with the tool, and how to use the tool to streamline and improve existing processes.
- To decide on standard ways of using the tool that will work for all potential users (for example naming conventions for files and tests, creation of libraries, defining modularity, where different elements will be stored, how they and the tool itself will be maintained, and coding standards for test scripts).

- To assess whether or not the benefits will be achieved at reasonable cost.
- To understand (and experiment with) metrics that you want the tool(s) to collect and to report, and configuring the tool(s) to ensure that your goals for these metrics can be achieved.

6.2.3 Success factors for tools

Success is not guaranteed or automatic when acquiring a testing tool, but many organizations have succeeded. After a successful selection process and a pilot project, two other things are also important to get the greatest benefit from the tools: the way in which the tool(s) is deployed within the wider organization, and the way in which ongoing support is organized. Here are some of the factors that contribute to success:

- Rolling out the tool incrementally (after the pilot) to the rest of the organization (a gradual uptake of the tool, not trying to get the whole organization to use it at once immediately).
- Adapting and improving processes, testware and tool artefacts to get the best fit and balance between them and the use of the tool.
- Providing adequate support, training (for example for those using the tool directly), coaching (for example from external specialist automation consultants) and mentoring for tool users.
- Defining and communicating guidelines for the use of the tool, based on what was learned in the pilot (for example internal standards for automation).
- Implementing a way to gather information about the use of the tool, to enable continuous improvement as tool use spreads through more of the organization.
- Monitoring the use of the tool and the benefits achieved, and adapting the use of the tool to take account of what is learned.
- Providing continuing support for anyone using test tools, such as the test team (for example, technical expertise is needed to help non-programmer testers who use keyword-driven testing).
- Gathering lessons learned, based on information gathered from all teams who are using test tools.

Any tools also need to be integrated both technically and organizationally into the software development life cycle. This may involve separate organizations that are responsible for different aspects, such as operations and/or third-party suppliers. Failure in tool use can come from any one factor; success needs all factors to be working together.

More information and advice about experiences in using tools can be found in Graham and Fewster [2012] and Gamba and Graham [2018].

CHAPTER REVIEW

Let's review what you have learned in this chapter.

From Section 6.1.1, you should now be able to classify different types of test tools according to the test process activities that they support. You should also recognize the tools that may help developers in their testing (shown by (D) below). In addition to the list below, you should recognize that there are tools that support specific application areas and that general purpose tools can also be used to support testing. The tools you should now recognize are:

Tools that support the management of testing and tests:

- Test management tools and application lifecycle management (ALM) tools.
- Requirements management tools.
- Defect management tools.
- Configuration management tools.
- Continuous integration tools (D).

Tools that support static testing:

- Tools that support reviews.
- Static analysis tools (D).

Tools that support test design and implementation:

- Test design tools.
- Model-Based Testing (MBT) tools.
- Test data preparation tools.
- Acceptance test-driven development (ATDD) and behaviour-driven development (BDD) tools.

Tools that support test execution and logging:

- Test execution tools.
- Coverage tools, for example requirements coverage, code coverage (D).
- Test harnesses (D).
- Unit test framework tools (D).

Tools that support performance measurement and dynamic analysis:

- Performance testing tools.
- Monitoring tools.
- Dynamic analysis tools (D).

You should know the Glossary terms **performance testing tool, test execution tool**, and **test management tool**.

From Section 6.1.2, you should be able to summarize the potential benefits and potential risks of test automation.

From Section 6.1.3, you should recognize the special considerations of test execution tools and test management tools.

From these two sections, you should know the Glossary terms **data-driven testing**, **keyword-driven testing** and **test automation**.

From Section 6.2.1, you should be able to state the main principles of introducing a tool into an organization.

From Section 6.2.2, you should be able to state the objectives of a pilot project as a starting point to introduce a tool into an organization.

From Section 6.2.3, you should recognize that simply acquiring a tool is not the only factor in achieving success in using test tools; there are many other factors that are important for success.

There are no specific definitions for these three sections.

SAMPLE EXAM QUESTIONS

Question 1 Which tools help to support static testing?

a. Static analysis tools and test execution tools.
b. Review tools, static analysis tools and coverage measurement tools.
c. Dynamic analysis tools and modelling tools.
d. Review tools and static analysis tools.

Question 2 Which test activities are supported by test harnesses?

a. Test management and control.
b. Test design and implementation.
c. Test execution and logging.
d. Performance measurement and dynamic analysis.

Question 3 What are the potential benefits from using test automation?

a. Greater quality of code, reduction in the number of testers needed, better objectives for testing.
b. Greater repeatability of tests, reduction in repetitive work, objective assessment.
c. Greater responsiveness of users, reduction of tests run, objectives not necessary.
d. Greater quality of code, reduction in paperwork, fewer objections to the tests.

Question 4 Which of the following are advanced scripting techniques for test execution tools?

a. Data-driven and keyword-driven.
b. Data-driven and capture-driven.
c. Capture-driven and keyhole-driven.
d. Playback-driven and keyword-driven.

Question 5 Which of the following would NOT be done as part of selecting a tool for an organization?

a. Assess organizational maturity, strengths and weaknesses.
b. Roll out the tool to as many users as possible within the organization.
c. Evaluate the tool features against clear requirements and objective criteria.
d. Identify internal requirements for coaching and mentoring in the use of the tool.

Question 6 Which of the following is a goal for a pilot project for introducing a tool into an organization?

a. Decide which tool to acquire.
b. Decide on the main objectives and requirements for this type of tool.
c. Evaluate the tool vendor including training, support, and commercial aspects.
d. Decide on standard ways of using, managing, storing and maintaining the tool and the test assets.

Question 7 What is a success factor for the use of tools within an organization?

a. Implement a way to gather usage information from the actual use of the tool.
b. Evaluate how the tool fits with existing processes and adopt the tool's approach.
c. Roll out the tool to all of the rest of the organization at once, as quickly as possible.
d. Assess whether the benefits will be achieved at reasonable cost.

CHAPTER SEVEN

ISTQB Foundation Exam

We wrote (and re-wrote and re-wrote) this book specifically to cover the International Software Testing Qualification Board (ISTQB) Foundation Syllabus 2018. Because of this, mastery of the previous six chapters should ensure that you master the exam too. Here are some thoughts on how to make sure you show how much you know when taking the exam.

7.1 PREPARING FOR THE EXAM

7.1.1 Studying for the exam

Whether you have taken a preparatory course along with reading this book or just read the book, we have some study tips for certification candidates who intend to take an exam under one of the ISTQB-recognized National Boards or Exam Boards.

Of course, you should carefully study the Syllabus. If you encounter any statements or concepts you do not *completely* understand, refer back to this book and to any course materials used, to prepare for the exam. Exam questions often turn on *precise* understanding of a concept or the wording.

While you should study the whole Syllabus, scrutinize the learning objectives in the Syllabus one by one. Ask yourself if you have achieved that learning objective to the given level of knowledge. If not, go back and review the appropriate material in the section corresponding to that learning objective. Note that the sections in both the Syllabus and the book often have the same numbering, and this is usually the same as the learning objective in the Syllabus.

Going beyond the Syllabus, notice that a number of international standards are referred to in the Syllabus. Some exam questions may be about the standards and there will also be questions about Glossary terms, and often even experienced testers are unfamiliar with them.

We recommend that you try to answer all of the sample exam questions in this book. If you have understood the material in a chapter, you should have no trouble with the questions. Make sure you understand why the right answer is the right answer and why the wrong answers are wrong. If you cannot answer a question and don't understand why, review that section.

We also recommend that you try to do all of the exercises in the book. If you can complete them correctly, you probably understand the concepts. If you do the exercises well, you are more likely to pass the exam.

Finally, be sure to take the mock exam we have provided at the end of this chapter. If you have understood the material in the book, you should have no trouble

with most, if not all, of the questions on the mock exam. Make sure you carefully review the book sections corresponding to any questions that you haven't answered correctly. When you take the mock exam, try to simulate exam conditions as much as possible. Set aside a full hour to do the exam, and make sure that you are not disturbed during the hour: no interruptions! Turn off email and leave your phone in another room.

7.1.2 The exam and the Syllabus

The ISTQB Foundation exam is designed to assess your knowledge and understanding of basic testing ideas and terms, which provides a solid starting point for your future testing efforts. These exams are not perfect. Even well-qualified candidates often fail a few questions or disagree with some answers. Study hard, so you can take the exam in a relaxed and confident way.

If you take the exam unprepared, you should expect your lack of preparation to show in your score. Since you will pay to take the exam, consider preparation time an investment in protecting your exam fee.

The ISTQB Foundation Syllabus 2018 is the current Syllabus. It replaces the 2011 version, which in turn replaced the 2007 version, which in turn replaced the 2005 version. Do not refer to the old Syllabi, as the exam covers the new Syllabus.

7.1.3 Where should you take the exam?

If you have taken an accredited training course, it is quite likely the exam will occur on the last day of the course. The exam fee may have been included in the course fee. If so, you should plan to study each evening during the course, including material that will be covered on the last day. Any topic covered in the Syllabus may be in the exam. You might even want to read this book before starting the course.

You might be taking an online course. Again, we recommend that you study the materials as you cover each section. Consider reading the book during or even before taking the course.

If you have studied on your own, without taking a course, then you will need to find a place open to the general public where the exam is offered. Many conferences around the world offer the exam this way. Otherwise the website for the testing board for your country (the National Board) should be able to help you find the right venue for a public exam.

In addition, ISTQB-recognized exam providers such as ASTQB, BCS (British Computer Society), Brightest, Cert-IT, CertInstitute, GASQ, iSQI and Kryterion (list adapted from ISTQB website at the time of publication) will provide you with ISTQB-branded certificates, as they work in close cooperation with the ISTQB and its National Boards. The ISTQB and each ISTQB-recognized National Board and Exam Board are constitutionally obliged to support a single, universally accepted, international qualification scheme. Therefore, you are free to choose ISTQB-accredited trainers, if desired, and ISTQB-accredited exam providers based on factors like convenience, competence, suitability of the training to your immediate professional and business needs and price.

7.2 TAKING THE EXAM

All the National Boards offer the same type of Foundation exam. It is a multiple choice exam that conforms to the guidelines established by the ISTQB. The exam in this book also follows the new guidelines from ISTQB for the balance of questions from the different sections/chapters. For some of us, it's been a while since we last took an exam and you may not remember all the tricks and techniques for taking a multiple choice exam. Here are a few paragraphs that will give you some ideas on how to take the exam.

7.2.1 How to approach multiple choice questions

Remember that there are two aspects to correctly answering a multiple choice exam question. First, you must understand the question and decide what the right answer is. Second, you must correctly communicate your answer by selecting the correct option from the choices listed.

Each question has one correct answer. This answer is always or most frequently true, given the stated context and conditions in the question. The other answers are intended to mislead people who do not completely understand the concept and are called 'distracters' in the examination business.

Read the question carefully and make sure you understand it before you decide on your answer. It may help to highlight or underline the keywords or concepts stated in the question. Some questions may be ambiguous or ask which is the best or worst alternative. You should choose the best answer based on what you have learned in this book. Remember that your own situation may be different from the most common situation or the ideal situation.

Some of the distracters may confuse you. In other words, you might be unsure about the correct answers to some of the questions. So make two passes through the exam. On your first pass, answer all of the questions you are sure of. Come back to the others later.

If a question will take a long time to answer, come back to it later, when you feel confident that you have enough time. Each correct answer is worth one point, whether it takes 30 seconds or 5 minutes to decide on it.

It's important to pace yourself. We suggest that you spend 45–60 seconds on each question. That way your first pass will take less than 40 minutes. You can then use the remaining time to have a one-minute break, then double-check your answers, and then finally go back to any questions you haven't answered yet.

If you use an answer sheet for the exam, they may vary slightly from one National Board or Exam Board to another. You should make sure that each answer corresponds to the correct question number. Especially if you have skipped a question; it is easy to get confused and mark the answer option above or below the question you are trying to answer.

Also, make sure you have an answer for every question. Double-check that you have selected the answer you want, as it is easy to select the wrong answer.

7.2.2 On trick questions

There are no deliberately designed trick questions in the ISTQB exams, but some can seem quite difficult if you are not completely sure of the right answer. Remember, just because it is hard to answer does not make it a trick. Here are some ideas for dealing with the tough questions.

First, read the question and each of the options very thoroughly. It is easy to read what you expect to be there rather than what is there.

If you are tempted to change your answer, do so only if you are quite sure. When you are undecided between two options, it's usually best to go with your first instinct. Remember that the correct answer is the one always or most often true. Simply because you can imagine a circumstance under which an answer might not be true does not make it wrong, just as knowing a 95-year-old cigarette smoker does not prove that cigarette smoking is not risky.

For most exams, you are allowed to write on the question paper or make notes on paper (which would need to be handed in). In addition, there is typically no penalty for the wrong answer. So feel free to guess!

While the ISTQB guidelines call for straightforward questions, some topics are difficult to cover without some amount of complexity. Be especially careful with negative questions and complex questions. If the question asks which is false (or true), mark them T or F first if you can. This will make it easier to see which is the odd one out. If the question requires matching a list of definitions, you might be able to draw lines between the definitions and the words they define. And remember to use the process of elimination to work for you whenever possible. Use what you know to eliminate wrong answers and cross them out up front to avoid mistakes.

Finally, remember that the ISTQB exam is about the theory of good practice, not what you typically do in practice. The ISTQB Syllabus was developed by an international working group of testing experts based on the good practices they have seen in the real world. In testing, good practices may lead typical practices by about 20 years' time. Some organizations are struggling to implement even basic testing principles, such as removing the misconception that "complete" testing is possible, while other organizations may already be doing most of the things discussed in the Syllabus. Still other organizations may do things well but differently from what is described in the Syllabus. So remember to apply what you have learned in this book to the exam. When an exam question calls into conflict what you have learned here and what you typically have done in the past, rely on what you have learned to answer the exam question. And then go back to your workplace and put the good ideas that you have learned into practice!

7.2.3 Last but not least

ISTQB has sample exam papers available to download, as well as the correct answers and justifications for those answers. It is well worth working through these exams as well! Search under 'Exams', 'Sample exams', and "Foundation level' for CTFL 2018 exams.

We wish you good luck with your certification exam and best of success in your career as a test professional! We stand poised on the brink of great forward progress in the field of testing and are happy that you will be a part of it.

7.3 MOCK EXAM

In the real exam, you will have 60 minutes (75 if the exam is not in your first language) to work through 40 questions of approximately the same difficulty mix and Syllabus distribution as shown in the following mock exam. After you have taken this mock exam, check your answers with the answer key. The answers to all exam questions in this book are in the section after the Glossary.

Question 1 Assume postal rates for light letters are:

$0.25 up to 10 grams.

$0.35 up to 50 grams.

$0.45 up to 75 grams.

$0.55 up to 100 grams.

Which test inputs (in grams) could be selected using equivalence partitioning?

a. 0, 9, 10, 49, 50, 74, 75, 99, 100.

b. 5, 35, 65, 95, 115.

c. 0, 1, 10, 11, 50, 51, 75, 76, 100, 101.

d. 5, 25, 35, 45, 55.

Question 2 Consider the following statements about testing and debugging and identify which are True:

1. Debugging is a testing activity.
2. Testers may be involved in debugging and component testing in Agile development.
3. Testing finds, analyzes and fixes defects.
4. Debugging executes tests to show failures caused by defects.
5. Checking whether fixes resolved the defects found is a form of testing.

a. 2 and 5.

b. 2, 3 and 4.

c. 1, 2 and 5.

d. Only 5 is True.

Question 3 Consider the following statements about the reasons to adapt life cycle models for specific projects or products:

I Different projects have different goals.

II The life cycle model should be adapted to suit all types of development within the company for consistency.

III Different types of product have different product risks.

IV Business priorities are different depending on the context of the project or product.

V Different test environments may be necessary.

Which of the statements are true?

a. I, III and IV.

b. I, II and IV.

c. III, IV and V.

d. II, IV and V.

Question 4 Which of the following statements about use case testing are True?

1. A use case actor may be a business user; a use case subject is the system.
2. Error messages are tested in the main behavioural flow of the use case.
3. Interactions may be represented by activity diagrams.
4. Coverage cannot be measured for use case testing.
5. Interactions may change the state of the subject.

a. 1, 2 and 3.

b. 1, 4 and 5.

c. 2, 3 and 4.

d. 1, 3 and 5.

Question 5 Consider the following list of either product or project risks:

I An incorrect calculation of fees might short-change the organization.

II A vendor might fail to deliver a system component on time.

III A defect might allow hackers to gain administrative privileges.

IV A skills gap might occur in a new technology used in the system.

V A defect-prioritization process might overload the development team.

Which of the following statements is true?

a. I is primarily a product risk and II, III, IV and V are primarily project risks.
b. II and V are primarily product risks and I, III and V are primarily project risks.
c. I and III are primarily product risks, while II, IV and V are primarily project risks.
d. III and V are primarily product risks, while I, II and IV are primarily project risks.

Question 6 Consider the following statements about regression tests:

I They may usefully be automated if they are well designed.
II They are the same as confirmation tests (re-tests).
III They are a way to reduce the risk of a change having an adverse affect elsewhere in the system.
IV They are only effective if automated.

Which pair of statements is true?

a. I and II.
b. I and III.
c. II and III.
d. II and IV.

Question 7 According to the ISTQB Glossary, what is a black-box test technique?

a. 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.
b. A procedure to derive and/or select test cases based on the tester's experience, knowledge and intuition.
c. A procedure to derive and/or select test cases based on an analysis of the specification, either functional or non-functional, of a component or system without reference to its internal structure.
d. A procedure to derive and/or select test cases based on an analysis of the internal structure of a component or system.

Question 8 Review the following portion of a defect report, which occurs after the defect ID, time and date, and author.

1. I place any item in the shopping cart.
2. I place any other (different) item in the shopping cart.
3. I remove the first item from the shopping cart, but leave the second item in the cart.
4. I click the <Checkout> button.
5. I expect the system to display the first checkout screen. Instead, it gives the pop-up error message, 'No items in shopping cart. Click <Okay> to continue shopping'.
6. I click <Okay>.
7. I expect the system to return to the main window to allow me to continue adding and removing items from the cart. Instead, the browser terminates.
8. The failure described in steps 5 and 7 occurred in each of three attempts to perform steps 1, 2, 3, 4 and 6.

Assume that no other narrative information is included in the report. Which of the following important aspects of a good defect report is missing from this report?

a. The steps to reproduce the failure.
b. The summary.
c. The check for intermittence.
d. The use of an objective tone.

Question 9 Which of the tasks below is typically done by a tester (not a test manager)?

a. Support setting up the defect management system.
b. Share testing perspectives with other project activities such as integration planning.
c. Design, set up and verify test environment(s).
d. Initiate the analysis, design, implementation and execution of tests.

Question 10 Consider the following factors:

1. The test levels and test types being considered.
2. The best practices implemented in other companies.
3. The business domain.
4. Required internal and external standards.
5. The way in which the previous system was developed.

Which of these factors may influence the test process in context?

a. 1, 2 and 5.

b. 2, 4 and 5.

c. 1, 3 and 4.

d. 1, 3 and 5.

Question 11 Of the following statements about reviews of specifications, which statement is true?

a. Reviews are not generally cost-effective as the meetings are time consuming and require preparation and follow up.

b. There is no need to prepare for or follow up on reviews.

c. Reviews must be controlled by the author.

d. Reviews are a cost-effective early static test on the system.

Question 12 Consider the following list of test process activities:

I Analysis and design.

II Test closure activities.

III Evaluating exit criteria and reporting.

IV Planning and control.

V Implementation and execution.

Which of the following places these in their logical sequence?

a. I, II, III, IV and V.

b. IV, I, V, III and II.

c. IV, I, V, II and III.

d. I, IV, V, III and II.

Question 13 Test objectives vary between projects and so should be stated in the test plan. Which one of the following test objectives might conflict with the proper tester mindset?

a. Show that the system works before we ship it.

b. Find as many defects as possible.

c. Reduce the overall level of product risk.

d. Prevent defects through early involvement.

Question 14 Which of the following is a test metric?

a. Percentage of planned functionality that has completed development.

b. Confirmation test results (pass/fail).

c. Cost of development, including time and effort.

d. Effort spent in fixing defects, not including confirmation testing or regression testing.

Question 15 If you are flying with an economy ticket, there is a possibility that you may get upgraded to business class, especially if you hold a gold card in the airline's frequent flyer programme. If you do not hold a gold card, there is a possibility that you will get bumped off the flight if it is full and you check in late. This is shown in Figure 7.1. Note that each box (i.e. statement) has been numbered.

Three tests have already been run:

Test 1: Gold card holder who gets upgraded to business class.

Test 2: Non-gold card holder who stays in economy.

Test 3: A person who is bumped from the flight.

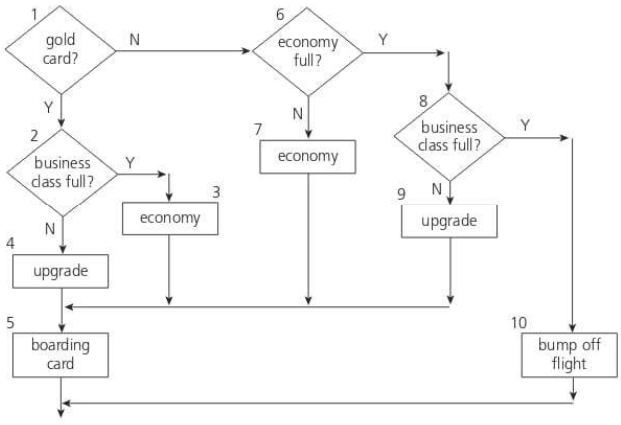

FIGURE 7.1 Control flow diagram for flight check-in

What additional tests would be needed to achieve 100% decision coverage?

a. A gold card holder who stays in economy and a non-gold card holder who gets upgraded to business class.

b. A gold card holder and a non-gold card holder who are both upgraded to business class.

c. A gold card holder and a non-gold card holder who both stay in economy class.

d. A gold card holder who is upgraded to business class and a non-gold card holder who stays in economy class.

Question 16 Consider the following types of tools:

V Test management tools.

W Static analysis tools.

X Continuous integration tools.

Y Dynamic analysis tools.

Z Performance testing tools.

Which of the following of these tools is most likely to be used by developers?

a. W, X and Y.

b. V, Y and Z.

c. V, W and Z.

d. X, Y and Z.

Question 17 Which of the following statements about error guessing are True?

P. Error guessing uses a test charter and test session sheets.

Q. Error guessing is based on tester knowledge.

R. Error guessing always uses a checklist.

S. Error guessing is a black-box test technique.

T. Error guessing can be based on mistakes developers may make.

a. P and T.

b. Q and R.

c. Q and T.

d. Q and S.

Question 18 Which of the following is the most important difference between the metrics-based approach and the expert-based approach to test estimation?

a. The metrics-based approach is more accurate than the expert-based approach.

b. The metrics-based approach uses calculations from historical data while the expert-based approach relies on team wisdom.

c. The metrics-based approach can be used to verify an estimate created using the expert-based approach, but not vice versa.

d. The expert-based approach takes longer than the metrics-based approach.

Question 19 If the temperature falls below 18 degrees, the heating is switched on. When the temperature reaches 21 degrees, the heating is switched off. What is the minimum set of test input values to cover all valid equivalence partitions?

a. 15, 19 and 25 degrees.

b. 17, 18, 20 and 21 degrees.

c. 18, 20 and 22 degrees.

d. 16 and 26 degrees.

Question 20 According to the ISTQB Glossary, what is coverage?

a. An attribute or combination of attributes that is derived from one or more test conditions by using a test technique that enables the measurement of the thoroughness of the test execution.

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

c. The degree to which specified tests have been run, compared to the full set of tests that could have been run.

d. The degree to which tests have been automated, compared to tests that could have been automated.

Question 21 Which of the following statements about static testing is False?

a. Reviews are applicable to web pages and user guides.

b. Static analysis is applicable to executing test scripts.

c. Static analysis can be applied to work products written in natural language (e.g. English).

d. Reviews can be applied to security requirements and project budgets.

Question 22 Which success factors are required for good tool support within an organization?

a. Acquiring the best tool and ensuring that all testers use it.

b. Adapting processes to fit with the use of the tool and monitoring tool use and benefits.

c. Setting ambitious objectives for tool benefits and aggressive deadlines for achieving them.
d. Adopting practices from other successful organizations and ensuring that initial ways of using the tool are maintained.

Question 23 Match the keyword terms to the correct definitions:

x. Something incorrect in a work product.
y. A person's mistake.
z. An event where the system does NOT perform correctly.

a. x. is a defect, y. is an error, z. is a failure.
b. x. is an error, y. is a defect, z. is a failure.
c. x. is a failure, y. is an error, z. is a defect.
d. x. is a defect, y. is a failure, z. is an error.

Question 24 Which of the following would be a typical defect found in component testing?

a. Incorrect sequencing or timing of interface calls.
b. Incorrect code and logic.
c. Business rules not implemented correctly.
d. Unhandled or improperly handled communication between components.

Question 25 Which of the following could be a root cause of a defect in financial software in which an incorrect interest rate is calculated?

a. Insufficient funds were available to pay the interest rate calculated.
b. Insufficient calculations of compound interest were included.
c. Insufficient training was given to the developers concerning compound interest calculation rules.
d. Inaccurate calculators were used to calculate the expected results.

Question 26 Assume postal rates for light letters are:

$0.25 up to 10 grams.
$0.35 up to 50 grams.
$0.45 up to 75 grams.
$0.55 up to 100 grams.

Which test inputs (in grams) would be selected using boundary value analysis?

a. 0, 9, 19, 49, 50, 74, 75, 99, 100.
b. 10, 50, 75, 100, 250, 1000.
c. 0, 1, 10, 11, 50, 51, 75, 76, 100, 101.
d. 25, 26, 35, 36, 45, 46, 55, 56.

Question 27 Consider the following decision table.

TABLE 7.1 Decision table for car rental

Conditions	Rule 1	Rule 2	Rule 3	Rule 4
Over 23?	F	T	T	T
Clean driving record?	Don't care	F	T	T
On business?	Don't care	Don't care	F	T
Actions				
Supply rental car?	F	F	T	T
Premium charge?	F	F	F	T

Given this decision table, what is the expected result for the following test cases?

TC1: A 26-year-old on business but with violations or accidents on his driving record.
TC2: A 62-year-old tourist with a clean driving record.

a. TC1: Do not supply car;
TC2: Supply car with premium charge.
b. TC1: Supply car with premium charge;
TC2: Supply car with no premium charge.
c. TC1: Do not supply car;
TC2: Supply car with no premium charge.
d. TC1: Supply car with premium charge;
TC2: Do not supply car.

Question 28 What is exploratory testing?

a. The process of anticipating or guessing where defects might occur.
b. A systematic approach to identifying specific equivalent classes of input.
c. The testing carried out by a chartered engineer.
d. Concurrent test design, test execution, test logging and learning.

Question 29 Which statement about functional, non-functional and white-box testing is True?

a. Functional testing evaluates characteristics such as reliability, security or usability; non-functional testing evaluates characteristics such as system architecture and the thoroughness of testing; white-box testing evaluates characteristics such as completeness and correctness.

b. Functional testing evaluates characteristics such as completeness and correctness; non-functional testing evaluates characteristics such as reliability, security or usability; white-box testing evaluates characteristics such as system architecture and the thoroughness of testing.

c. Functional testing evaluates characteristics such as completeness and correctness; non-functional testing evaluates characteristics such as system architecture and the thoroughness of testing; white-box testing evaluates characteristics such as reliability, security or usability.

d. Functional testing evaluates characteristics such as system architecture and the thoroughness of testing; non-functional testing evaluates characteristics such as reliability, security or usability; white-box testing evaluates characteristics such as completeness and correctness.

Question 30 A test plan is written specifically to describe a level of testing where the primary goal is establishing confidence in the system. Which of the following is a likely name for this document?

a. Master test plan.

b. System test plan.

c. Acceptance test plan.

d. Project plan.

Question 31 How do experience-based test techniques differ from black-box test techniques?

a. They depend on the tester's understanding of the way the system is structured rather than on a documented record of what the system should do.

b. They depend on an individual's domain knowledge and expertise rather than on a documented record of what the system should do.

c. They depend on a documented record of what the system should do rather than on an individual's personal view.

d. They depend on having older testers rather than younger testers.

Question 32 Which of the following statements are characteristics of static testing, and which are characteristics of dynamic testing?

1. Finds defects in work products directly.
2. Better for ensuring internal quality (e.g. standards are followed).
3. Finds defects through failures in execution.
4. Easier and cheaper to find and fix security vulnerabilities (e.g. buffer overflow).
5. Focuses on externally visible behaviours.

a. Static testing: 2 and 4, Dynamic testing: 1, 3 and 5.

b. Static testing: 3 and 5, Dynamic testing: 1, 2 and 4.

c. Static testing: 1, 2 and 4, Dynamic testing: 3 and 5.

d. Static testing: 1, 2 and 3, Dynamic testing: 4 and 5.

Question 33 System test execution on a project is planned for eight weeks. After a week of testing, a tester suggests that the test objective stated in the test plan of 'finding as many defects as possible during system test' might be more closely met by redirecting the test effort in what way?

a. By asking a selection of users what is most important for them, and testing that.

b. By testing the main workflows of the business.

c. By repeating the unit and integration tests.

d. By testing in areas where the most defects have already been found.

Question 34 Consider the following activities that might relate to configuration management:

I Identify and document the characteristics of a test item.

II Control changes to the characteristics of a test item.

III Check a test item for defects introduced by a change.

IV Record and report the status of changes to test items.

V Confirm that changes to a test item fixed a defect.

Which of the following statements is True?

a. Only I is a configuration management task.
b. All are configuration management tasks.
c. I, II and III are configuration management tasks.
d. I, II and IV are configuration management tasks.

Question 35 Consider the following state transition diagram.

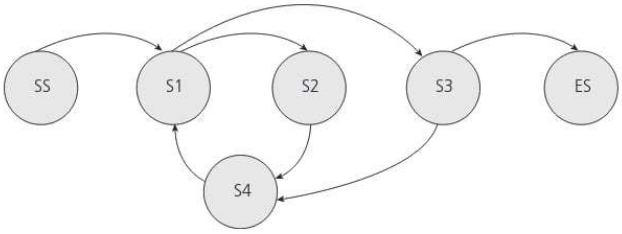

FIGURE 7.2 State transition diagram

Given this diagram, which test case below covers every valid transition?

a. SS – S1 – S2 – S4 – S1 – S3 – ES
b. SS – S1 – S2 – S3 – S4 – S3 – S4 – ES
c. SS – S1 – S2 – S4 – S1 – S3 – S4 – S1 – S3 – ES
d. SS – S1 – S4 – S2 – S1 – S3 – ES

Question 36

You are the test manager for a user acceptance test project. You have decided to use a requirements-based and risk-based test strategy. Test suites have been designed to cover all of the requirements.

Table 7.2 (below) shows the priority and the designed test suites associated with the requirements (a lower priority number means higher risk).

The table also shows the configuration of the test environment in which the requirements can be tested. Thus two different configurations, ABC and XYZ, are required to execute all test suites.

Assume that the test environment is delivered initially to the test team under the configuration ABC; XYZ will be available from the start of week 4. Each test suite will take approximately 1 week to execute.

The execution of TS4 depends on TS3 to be executed before, and the execution of TS2 depends on TS1 to be executed before.

Based only on the given information, which of the following test execution schedules would be preferred given the constraints of the test environment and the test strategy chosen?

a. TS5, TS6, TS4, TS3, TS1, TS2, TS7.
b. TS5, TS4, TS1, TS6, TS3, TS2, TS7.
c. TS5, TS6, TS1, TS3, TS4, TS2, TS7.
d. TS5, TS6, TS1, TS2, TS7, TS4, TS3.

TABLE 7.2 Priority and dependency table for Question 36

Requirement	*Priority*	*Test Suites*	*Environment*	*Dependency*
R1	4	TS1	ABC	
R2	4	TS2	ABC	TS1
R3	3	TS3	XYZ	
R4	2	TS4	XYZ	TS3
R5	1	TS5, TS6	ABC	
R6	5	TS7	ABC	

Question 37 What is the most important factor for successful performance of reviews?

a. A separate scribe during the logging meeting.

b. Trained participants and review leaders.

c. The availability of tools to support the review process.

d. A reviewed test plan.

Question 38 How is impact analysis used in maintenance testing?

a. It evaluates intended consequences and possible side effects of a change to the system, in order to plan what testing to do.

b. It evaluates the consequences of an intended change on the test environment.

c. It evaluates the ease of maintaining the system when there are changes to the system or to the tests.

d. It evaluates the existing regression tests to assess whether a planned change to the system should go ahead.

Question 39 When you sign up, you must give your first and last name, address and postcode or zip code, your mobile/cell phone number, your email address and set yourself a password.

When you log in, you must give the following details: last name, phone number and password. You are logged in until you select 'Logout' followed by answering 'Yes' to 'Are you sure?'

You can update your details once you are logged in, but you need to confirm the change by entering a code sent to your phone. You then need to log in again.

In reviewing the specification above, which of the following are potential defects likely to be found by a user perspective review?

1. Cannot log back in if not logged yourself out (e.g. when changing details).
2. Incorrect format of the code sent to the phone.
3. Buffer overflow for a postal address that is too long.
4. Cannot change the phone number, as the code is sent to the old phone.
5. When details are changed, they create a separate new record in the database.

a. 1 and 2.

b. 3 and 5.

c. 1 and 4.

d. 4 and 5.

Question 40 Which of the following is an advantage of independent testing?

a. Independent testers do NOT have to spend time communicating with the project team.

b. Developers can stop worrying about the quality of their work and focus on producing more code.

c. The others on a project can pressure the independent testers to accelerate testing at the end of the schedule.

d. Independent testers sometimes question the assumptions behind requirements, designs and implementations.

GLOSSARY

This glossary provides the complete definition set of software testing terms, as defined by ISTQB.

The glossary has been arranged in a single section of definitions ordered alphabetically. Some terms are preferred to other synonymous ones, in which case the definition of the preferred term appears, with the synonyms listed afterwards. For example, a synonym of *white-box testing* is *structural testing.*

'See also' cross-references are also used. They assist the user to quickly navigate to the right index term. 'See also' cross-references are constructed for relationships such as broader term to a narrower term and overlapping meanings between two terms.

Finally, note that the terms that are underlined are those that are specifically mentioned as keywords in the Syllabus at the beginning of Chapters 1 to 6. These are the terms that you should know for the exam.

Acceptance criteria The criteria that a component or system must satisfy in order to be accepted by a user, customer or other authorized entity.

Acceptance testing Formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system. See also: *user acceptance testing.*

Accessibility The degree to which a component or system can be used by people with the widest range of characteristics and capabilities to achieve a specified goal in a specified context of use.

Accessibility testing Testing to determine the ease by which users with disabilities can use a component or system.

Actual result The behavior produced/observed when a component or system is tested. Synonym: actual outcome

Ad hoc reviewing A review technique carried out by independent reviewers informally, without a structured process.

Alpha testing Simulated or actual operational testing conducted in the developer's test environment, by roles outside the development organization.

Anomaly Any condition that deviates from expectation based on requirements specifications, design documents, user documents, standards, etc., or from someone's perception or experience. Anomalies may be found during, but not limited to, reviewing, testing, analysis, compilation, or use of software products or applicable documentation.

Audit An independent examination of a work product, process or set of processes that is performed by a third party to assess compliance with specifications, standards, contractual agreements or other criteria.

Availability The degree to which a component or system is operational and accessible when required for use.

Behavior (behaviour) The response of a component or system to a set of input values and preconditions.

Beta testing Simulated or actual operational testing conducted at an external site, by roles outside the development organization. Synonym: field testing

Black-box test technique A procedure to derive and/or select test cases based on an analysis of the specification, either functional or non-functional, of a component or system without reference to its internal structure. Synonyms: black-box technique, specification-based technique, specification-based test technique

Boundary value A minimum or maximum value of an ordered equivalence partition.

Boundary value analysis A black-box test technique in which test cases are designed based on boundary values. See also: *boundary value.*

Burndown chart A publicly displayed chart that depicts the outstanding effort versus time in an iteration. It shows the status and trend of completing the tasks of the iteration. The X-axis

typically represents days in the sprint, while the Y-axis is the remaining effort (usually either in ideal engineering hours or story points).

Checklist-based reviewing A review technique guided by a list of questions or required attributes.

Checklist-based testing An experience-based test technique whereby the experienced tester uses a high-level list of items to be noted, checked, or remembered, or a set of rules or criteria against which a product has to be verified.

Code coverage An analysis method that determines which parts of the software have been executed (covered) by the test suite and which parts have not been executed, for example, statement coverage, decision coverage or condition coverage.

Commercial off-the-shelf (COTS) A software product that is developed for the general market, i.e. for a large number of customers, and that is delivered to many customers in identical format.
Synonym: off-the-shelf software

Compatibility The degree to which a component or system can exchange information with other components or systems.

Complexity The degree to which a component or system has a design and/or internal structure that is difficult to understand, maintain and verify.

Compliance The capability of the software product to adhere to standards, conventions or regulations in laws and similar prescriptions.

Component A minimal part of a system that can be tested in isolation.
Synonyms: module, unit

Component integration testing Testing performed to expose defects in the interfaces and interactions between integrated components.
Synonym: link testing

Component specification A description of a component's function in terms of its output values for specified input values under specified conditions, and required non-functional behavior (for example, resource utilization).

Component testing The testing of individual hardware or software components.
Synonyms: module testing, unit testing

Condition A logical expression that can be evaluated as True or False, for example, A>B.
Synonym: branch condition

Configuration The composition of a component or system as defined by the number, nature and interconnections of its constituent parts.

Configuration item An aggregation of work products that is designated for configuration management and treated as a single entity in the configuration management process.

Configuration management A discipline applying technical and administrative direction and surveillance to identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements.

Configuration management tool A tool that provides support for the identification and control of configuration items, their status over changes and versions, and the release of baselines consisting of configuration items.

Confirmation testing Dynamic testing conducted after fixing defects with the objective to confirm that failures caused by those defects do not occur anymore.
Synonym: re-testing

Contractual acceptance testing Acceptance testing conducted to verify whether a system satisfies its contractual requirements.

Control flow The sequence in which operations are performed during the execution of a test item.

Cost of quality The total costs incurred on quality activities and issues and often split into prevention costs, appraisal costs, internal failure costs and external failure costs.

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

Coverage item An attribute or combination of attributes that is derived from one or more test conditions by using a test technique that enables the measurement of the thoroughness of the test execution.

Coverage tool A tool that provides objective measures of what structural elements, for example, statements, branches have been exercised by a test suite.
Synonym: coverage measurement tool

Data flow An abstract representation of the sequence and possible changes of the state of

data objects, where the state of an object is any of creation, usage or destruction.

Data-driven testing A scripting technique that stores test input and expected results in a table or spreadsheet, so that a single control script can execute all of the tests in the table. Data-driven testing is often used to support the application of test execution tools such as capture/playback tools. See also: *keyword-driven testing.*

Debugging The process of finding, analyzing and removing the causes of failures in software.

Decision A type of statement in which a choice between two or more possible outcomes controls which set of actions will result.

Decision coverage The coverage of decision outcomes.

Decision outcome The result of a decision that determines the next statement to be executed.

Decision table A table used to show sets of conditions and the actions resulting from them.
Synonym: cause-effect decision table

Decision table testing A black-box test technique in which test cases are designed to execute the combinations of inputs and/or stimuli (causes) shown in a decision table. See also: *decision table.*

Decision testing A white-box test technique in which test cases are designed to execute decision outcomes.

Defect An imperfection or deficiency in a work product where it does not meet its requirements or specifications.
Synonyms: bug, fault

Defect density The number of defects per unit size of a work product.
Synonym: fault density

Defect management The process of recognizing and recording defects, classifying them, investigating them, taking action to resolve them and disposing of them when resolved.

Defect management tool A tool that facilitates the recording and status tracking of defects.
Synonyms: bug tracking tool, defect tracking tool

Defect report Documentation of the occurrence, nature and status of a defect. See also: *incident report.*
Synonym: bug report

Driver A software component or test tool that replaces a component that takes care of the control and/or the calling of a component or system.
Synonym: test driver

Dynamic analysis The process of evaluating behavior, for example, memory performance, CPU usage, of a system or component during execution.

Dynamic analysis tool A tool that provides run-time information on the state of the software code. These tools are most commonly used to identify unassigned pointers, check pointer arithmetic and to monitor the allocation, use and de-allocation of memory and to flag memory leaks.

Dynamic testing Testing that involves the execution of the software of a component or system.

Effectiveness Extent to which correct and complete goals are achieved. See also: *efficiency.*

Efficiency Resources expended in relation to the extent with which users achieve specified goals. See also: *effectiveness.*

Entry criteria The set of conditions for officially starting a defined task.
Synonym: definition of ready

Equivalence partition A portion of the value domain of a data element related to the test object for which all values are expected to be treated the same based on the specification.
Synonym: equivalence class

Equivalence partitioning A black-box test technique in which test cases are designed to exercise equivalence partitions by using one representative member of each partition.
Synonym: partition testing

Error A human action that produces an incorrect result.
Synonym: mistake

Error guessing A test technique in which tests are derived on the basis of the tester's knowledge of past failures, or general knowledge of failure modes.

Executable statement A statement which, when compiled, is translated into object code, and which will be executed procedurally when the program is running and may perform an action on data.

Exercised A program element is said to be exercised by a test case when the input value causes the execution of that element, such as a statement, decision or other structural element.

Exhaustive testing A test approach in which the test suite comprises all combinations of input values and preconditions.
Synonym: complete testing

Exit criteria The set of conditions for officially completing a defined task.
Synonyms: completion criteria, test completion criteria, definition of done

Expected result The predicted observable behavior of a component or system executing under specified conditions, based on its specification or another source.
Synonyms: expected outcome, predicted outcome

Experience-based test technique A procedure to derive and/or select test cases based on the tester's experience, knowledge and intuition.
Synonym: experience-based technique

Experience-based testing Testing based on the tester's experience, knowledge and intuition.

Exploratory testing An approach to testing whereby the testers dynamically design and execute tests based on their knowledge, exploration of the test item and the results of previous tests.

Extreme Programming (XP) A software engineering methodology used within Agile software development where core practices are programming in pairs, doing extensive code review, unit testing of all code, and simplicity and clarity in code. See also: *Agile software development.*

Facilitator The leader and main person responsible for an inspection or review process. See also: *moderator.*

Fail A test is deemed to fail if its actual result does not match its expected result.

Failure An event in which a component or system does not perform a required function within specified limits.

Failure rate The ratio of the number of failures of a given category to a given unit of measure.

Feature An attribute of a component or system specified or implied by requirements documentation (for example, reliability, usability or design constraints).
Synonym: software feature

Finding A result of an evaluation that identifies some important issue, problem or opportunity.

Formal review A form of review that follows a defined process with a formally documented output.

Functional integration An integration approach that combines the components or systems for the purpose of getting a basic functionality working early. See also: *integration testing.*

Functional requirement A requirement that specifies a function that a component or system must be able to perform.

Functional suitability The degree to which a component or system provides functions that meet stated and implied needs when used under specified conditions.
Synonym: functionality

Functional testing Testing conducted to evaluate the compliance of a component or system with functional requirements. See also: *black-box test technique.*

GUI Acronym for Graphical User Interface.

High-level test case A test case without concrete values for input data and expected results. See also: *low-level test case.*
Synonyms: abstract test case, logical test case

IDEAL An organizational improvement model that serves as a roadmap for initiating, planning and implementing improvement actions. The IDEAL model is named for the five phases it describes: initiating, diagnosing, establishing, acting and learning.

Impact analysis The identification of all work products affected by a change, including an estimate of the resources needed to accomplish the change.

Incident report Documentation of the occurrence, nature and status of an incident.
Synonyms: deviation report, software test incident report, test incident report. See also: *defect report*

Incremental development model A development life cycle model in which the project scope is generally determined early in the project life cycle, but time and cost estimates are routinely modified as the project team understanding of the product increases. The product is developed through a series of repeated cycles, each delivering an increment which successively adds to the functionality of the product. See also: *iterative development model.*

Independence of testing Separation of responsibilities, which encourages the accomplishment of objective testing.

Informal group review An informal review performed by three or more persons. See also: *informal review.*

Informal review A type of review without a formal (documented) procedure.

Input Data received by a component or system from an external source.

Inspection A type of formal review to identify issues in a work product, which provides measurement to improve the review process and the software development process.

Installation guide Supplied instructions on any suitable media, which guides the installer through the installation process. This may be a manual guide, step-by-step procedure, installation wizard or any other similar process description.

Integration The process of combining components or systems into larger assemblies.

Integration testing Testing performed to expose defects in the interfaces and in the interactions between integrated components or systems. See also: *component integration testing, system integration testing.*

Interoperability The degree to which two or more components or systems can exchange information and use the information that has been exchanged.

Interoperability testing Testing to determine the interoperability of a software product. See also: *functionality testing.*
Synonym: compatibility testing

Iterative development model A development life cycle where a project is broken into a usually large number of iterations. An iteration is a complete development loop resulting in a release (internal or external) of an executable product, a subset of the final product under development, which grows from iteration to iteration to become the final product.

Keyword-driven testing A scripting technique that uses data files to contain not only test data and expected results but also keywords related to the application being tested. The keywords are interpreted by special supporting scripts that are called by the control script for the test. See also: *data-driven testing.*
Synonym: action word-driven testing

Life cycle model A description of the processes, workflows and activities used in the development, delivery, maintenance and retirement of a system. See also: *software life cycle.*

Load testing A type of performance testing conducted to evaluate the behavior of a component or system under varying loads, usually between anticipated conditions of low, typical and peak usage. See also: *performance testing, stress testing.*

Low-level test case A test case with concrete values for input data and expected results. See also: *high-level test case.*
Synonym: concrete test case

Maintainability The degree to which a component or system can be modified by the intended maintainers.

Maintenance The process of modifying a component or system after delivery to correct defects, improve quality attributes or adapt to a changed environment.

Maintenance testing Testing the changes to an operational system or the impact of a changed environment to an operational system.

Master test plan A test plan that is used to coordinate multiple test levels or test types. See also: *test plan.*

Maturity (1) The capability of an organization with respect to the effectiveness and efficiency of its processes and work practices. (2) The degree to which a component or system meets needs for reliability under normal operation.

Measure The number or category assigned to an attribute of an entity by making a measurement.

Measurement The process of assigning a number or category to an entity to describe an attribute of that entity.

Memory leak A memory access failure due to a defect in a program's dynamic store allocation logic that causes it to fail to release memory after it has finished using it, eventually causing the program and/or other concurrent processes to fail due to lack of memory.

Metric A measurement scale and the method used for measurement.

Milestone A point in time in a project at which defined (intermediate) deliverables and results should be ready.

Model-based testing (MBT) Testing based on or involving models.

Moderator A neutral person who conducts a usability test session. See also: *facilitator*. Synonym: inspection leader

Monitoring tool A software tool or hardware device that runs concurrently with the component or system under test and supervises, records and/or analyzes the behavior of the component or system. See also: *dynamic analysis tool*.

Non-functional requirement A requirement that describes how well the component or system will do what it is intended to do.

Non-functional testing Testing conducted to evaluate the compliance of a component or system with non-functional requirements.

Operational acceptance testing Operational testing in the acceptance test phase, typically performed in a (simulated) operational environment by operations and/or systems administration staff focusing on operational aspects, for example, recoverability, resource-behavior, installability and technical compliance. See also: *operational testing*. Synonym: production acceptance testing

Operational environment Hardware and software products installed at users' or customers' sites where the component or system under test will be used. The software may include operating systems, database management systems and other applications.

Output Data transmitted by a component or system to an external destination.

Pass A test is deemed to pass if its actual result matches its expected result.

Path A sequence of events, for example, executable statements, of a component or system from an entry point to an exit point. Synonym: control flow path

Peer review A form of review of work products performed by others qualified to do the same work.

Performance efficiency The degree to which a component or system uses time, resources and capacity when accomplishing its designated functions. Synonyms: time behavior, performance

Performance indicator A high-level metric of effectiveness and/or efficiency used to guide and control progressive development, for example, lead-time slip for software development. Synonym: key performance indicator

Performance testing Testing to determine the performance of a software product.

Performance testing tool A test tool that generates load for a designated test item and that measures and records its performance during test execution.

Perspective-based reading A review technique whereby reviewers evaluate the work product from different viewpoints. See also: *role-based reviewing*.

Planning poker A consensus-based estimation technique, mostly used to estimate effort or relative size of user stories in Agile software development. It is a variation of the Wideband Delphi method using a deck of cards with values representing the units in which the team estimates. See also: *Agile software development, Wideband Delphi*.

Portability The ease with which the software product can be transferred from one hardware or software environment to another.

Portability testing Testing to determine the portability of a software product. Synonym: configuration testing

Postcondition The expected state of a test item and its environment at the end of test case execution.

Precondition The required state of a test item and its environment prior to test case execution.

Priority The level of (business) importance assigned to an item, for example, a defect.

Probe effect The effect on the component or system by the measurement instrument when the component or system is being measured, for example, by a performance testing tool or monitor. For example, performance may be slightly worse when performance testing tools are being used.

Problem An unknown underlying cause of one or more incidents.

Process A set of interrelated activities, which transform inputs into outputs.

Process improvement A program of activities designed to improve the performance and maturity of the organization's processes, and the result of such a program.

Product risk A risk impacting the quality of a product. See also: *risk*.

Project A project is a unique set of coordinated and controlled activities with start and finish dates undertaken to achieve an objective conforming to specific requirements, including the constraints of time, cost and resources.

Project risk A risk that impacts project success. See also: *risk*.

Quality The degree to which a component, system or process meets specified requirements and/or user/customer needs and expectations.

Quality assurance Part of quality management focused on providing confidence that quality requirements will be fulfilled.

Quality characteristic A category of product attributes that bears on quality.
Synonyms: software product characteristic, software quality characteristic, quality attribute

Quality control The operational techniques and activities, part of quality management, that are focused on fulfilling quality requirements.

Quality management Coordinated activities to direct and control an organization with regard to quality. Direction and control with regard to quality generally includes the establishment of the quality policy and quality objectives, quality planning, quality control, quality assurance and quality improvement.

Quality risk A product risk related to a quality characteristic. See also: *quality characteristic*, *product risk*.

Rational Unified Process (RUP) A proprietary adaptable iterative software development process framework consisting of four project life cycle phases: inception, elaboration, construction and transition.

Regression A degradation in the quality of a component or system due to a change.

Regression testing Testing of a previously tested component or system following modification to ensure that defects have not been introduced or have been uncovered in unchanged areas of the software as a result of the changes made.

Regulatory acceptance testing Acceptance testing conducted to verify whether a system conforms to relevant laws, policies and regulations.

Reliability The degree to which a component or system performs specified functions under specified conditions for a specified period of time.

Reliability growth model A model that shows the growth in reliability over time during continuous testing of a component or system as a result of the removal of defects that result in reliability failures.

Requirement A provision that contains criteria to be fulfilled.

Requirements management tool A tool that supports the recording of requirements, requirements attributes (for example, priority, knowledge responsible) and annotation, and facilitates traceability through layers of requirements and requirements change management. Some requirements management tools also provide facilities for static analysis, such as consistency checking and violations to pre-defined requirements rules.

Result The consequence/outcome of the execution of a test. It includes outputs to screens, changes to data, reports and communication messages sent out. See also: *actual result*, *expected result*.
Synonyms: outcome, test outcome, test result

Retrospective meeting A meeting at the end of a project during which the project team members evaluate the project and learn lessons that can be applied to the next project.
Synonym: post-project meeting

Review A type of static testing during which a work product or process is evaluated by one or more individuals to detect issues and to provide improvements.

Review plan A document describing the approach, resources and schedule of intended review activities. It identifies, amongst others: documents and code to be reviewed, review types to be used, participants, as well as entry and exit criteria to be applied in case of formal reviews, and the rationale for their choice. It is a record of the review planning process.

Reviewer A participant in a review, who identifies issues in the work product.
Synonyms: checker, inspector

Risk A factor that could result in future negative consequences.

Risk analysis The overall process of risk identification and risk assessment.

Risk-based testing Testing in which the management, selection, prioritization and use of testing activities and resources are based on corresponding risk types and risk levels.

Risk level The qualitative or quantitative measure of a risk defined by impact and likelihood. Synonym: risk exposure

Risk management The coordinated activities to direct and control an organization with regard to risk.

Risk mitigation The process through which decisions are reached and protective measures are implemented for reducing or maintaining risks to specified levels. Synonym: risk control

Risk type A set of risks grouped by one or more common factors. Synonym: risk category

Robustness The degree to which a component or system can function correctly in the presence of invalid inputs or stressful environmental conditions. See also: *error-tolerance, fault-tolerance.*

Role-based reviewing A review technique where reviewers evaluate a work product from the perspective of different stakeholder roles.

Root cause A source of a defect such that if it is removed, the occurrence of the defect type is decreased or removed.

Root cause analysis An analysis technique aimed at identifying the root causes of defects. By directing corrective measures at root causes, it is hoped that the likelihood of defect recurrence will be minimized. Synonym: causal analysis

Safety The capability that a system will not, under defined conditions, lead to a state in which human life, health, property or the environment is endangered.

Scenario-based reviewing A review technique where the review is guided by determining the ability of the work product to address specific scenarios.

Scribe A person who records information during the review meetings. Synonym: recorder

Scrum An iterative incremental framework for managing projects commonly used with Agile software development. See also: *Agile software development.*

Security The degree to which a component or system protects information and data so that persons or other components or systems have the degree of access appropriate to their types and levels of authorization.

Security testing Testing to determine the security of the software product. See also: *functionality testing.*

Sequential development model A type of development life cycle model in which a complete system is developed in a linear way of several discrete and successive phases with no overlap between them.

Session-based testing An approach to testing in which test activities are planned as uninterrupted sessions of test design and execution, often used in conjunction with exploratory testing.

Severity The degree of impact that a defect has on the development or operation of a component or system.

Simulation The representation of selected behavioral characteristics of one physical or abstract system by another system.

Simulator A device, computer program or system used during testing, which behaves or operates like a given system when provided with a set of controlled inputs. See also: *emulator.*

Software Computer programs, procedures and possibly associated documentation and data pertaining to the operation of a computer system.

Software development life cycle The activities performed at each stage in software development, and how they relate to one another logically and chronologically.

Software life cycle The period of time that begins when a software product is conceived and ends when the software is no longer available for use. The software life cycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase and sometimes, retirement phase. Note these phases may overlap or be performed iteratively.

Software quality The totality of functionality and features of a software product that bear on its ability to satisfy stated or implied needs. See also: *quality.*

Specification A document that specifies, ideally in a complete, precise and verifiable manner, the requirements, design, behavior, or other characteristics of a component or system, and, often, the procedures for determining whether these provisions have been satisfied.

Stability The degree to which a component or system can be effectively and efficiently modified without introducing defects or degrading existing product quality.

Standard Formal, possibly mandatory, set of requirements developed and used to prescribe consistent approaches to the way of working or to provide guidelines (for example, ISO/IEC standards, IEEE standards and organizational standards).

State diagram A diagram that depicts the states that a component or system can assume, and shows the events or circumstances that cause and/or result from a change from one state to another.
Synonym: state transition diagram

State transition A transition between two states of a component or system.

State transition testing A black-box test technique using a state transition diagram or state table to derive test cases to evaluate whether the test item successfully executes valid transitions and blocks invalid transitions. See also: *N-switch testing.*
Synonym: finite state testing

Statement An entity in a programming language, which is typically the smallest indivisible unit of execution.
Synonym: source statement

Statement coverage The percentage of executable statements that have been exercised by a test suite.

Statement testing A white-box test technique in which test cases are designed to execute statements.

Static analysis The process of evaluating a component or system without executing it, based on its form, structure, content or documentation.

Static testing Testing a work product without code being executed.

Structural coverage Coverage measures based on the internal structure of a component or system.

Stub A skeletal or special purpose implementation of a software component, used to develop or test a component that calls or is otherwise dependent on it. It replaces a called component.

System A collection of interacting elements organized to accomplish a specific function or set of functions.

System integration testing Testing the combination and interaction of systems.

System testing Testing an integrated system to verify that it meets specified requirements.

System under test (SUT) A type of test object that is a system.

Technical review A formal review type by a team of technically-qualified personnel that examines the suitability of a work product for its intended use and identifies discrepancies from specifications and standards.

Test A set of one or more test cases.

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

Test approach The implementation of the test strategy for a specific project.

Test automation The use of software to perform or support test activities, for example, test management, test design, test execution and results checking.

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

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

Test case specification Documentation of a set of one or more test cases.

Test charter Documentation of test activities in session-based exploratory testing. See also: *exploratory testing.*
Synonym: charter

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 condition An aspect of the test basis that is relevant in order to achieve specific test objectives.

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.
See also: *test management.*

Test cycle Execution of the test process against a single identifiable release of the test object.

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

Test data preparation tool A type of test tool that enables data to be selected from existing databases or created, generated, manipulated and edited for use in testing.
Synonym: test generator

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

Test design tool A tool that supports the test design activity by generating test inputs from a specification that may be held in a CASE tool repository, e.g., requirements management tool, from specified test conditions held in the tool itself, or from code.

Test environment An environment containing hardware, instrumentation, simulators, software tools and other support elements needed to conduct a test.
Synonyms: test bed, test rig

Test estimation The calculated approximation of a result related to various aspects of testing (for example, effort spent, completion date, costs involved, number of test cases, etc.), which is usable even if input data may be incomplete, uncertain or noisy.

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

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

Test execution tool A test tool that executes tests against a designated test item and evaluates the outcomes against expected results and postconditions.

Test harness A test environment comprised of stubs and drivers needed to execute a test.

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

Test infrastructure The organizational artefacts needed to perform testing, consisting of test environments, test tools, office environment and procedures.

Test input The data received from an external source by the test object during test execution. The external source can be hardware, software or human.

Test item A part of a test object used in the test process.
See also: *test object*

Test leader On large projects, the person who reports to the test manager and is responsible for project management of a particular test level or a particular set of testing activities. See also: *test manager*.
Synonym: lead tester

Test level A specific instantiation of a test process.
Synonym: test stage

Test management The planning, scheduling, estimating, monitoring, reporting, control and completion of test activities.

Test management tool A tool that provides support to the test management and control part of a test process. It often has several capabilities, such as testware management, scheduling of tests, the logging of results, progress tracking, incident management and test reporting.

Test manager The person responsible for project management of testing activities and resources, and evaluation of a test object. The individual who directs, controls, administers, plans and regulates the evaluation of a test object.

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. See also: *test management*.

Test object The component or system to be tested. See also: *test item*.

Test objective A reason or purpose for designing and executing a test.

Test oracle A source to determine expected results to compare with the actual result of the system under test.
Synonym: oracle

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

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

Test policy A high-level document describing the principles, approach and major objectives of the organization regarding testing.
Synonym: organizational test policy

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.
See also: *test script*.

Test process The set of interrelated activities comprising of test planning, test monitoring and control, test analysis, test design, test implementation, test execution and test completion.

Test process improvement A program of activities designed to improve the performance and maturity of the organization's test processes and the results of such a program.

Test progress report A test report produced at regular intervals about the progress of test activities against a baseline, risks, and alternatives requiring a decision.
Synonym: test status report

Test report Documentation summarizing test activities and results.

Test reporting Collecting and analyzing data from testing activities and subsequently consolidating the data in a report to inform stakeholders. See also: *test process*.

Test schedule A list of activities, tasks or events of the test process, identifying their intended start and finish dates and/or times, and interdependencies.

Test script A sequence of instructions for the execution of a test. See also: *test procedure*.

Test session An uninterrupted period of time spent in executing tests. In exploratory testing, each test session is focused on a charter, but testers can also explore new opportunities or issues during a session. The tester creates and executes on the fly and records their progress. See also: *exploratory testing*.

Test strategy Documentation that expresses the generic requirements for testing one or more projects run within an organization, providing detail on how testing is to be performed, and is aligned with the test policy.
Synonym: organizational test strategy

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

Test summary report A test report that provides an evaluation of the corresponding test items against exit criteria.
Synonym: test report

Test technique A procedure used to derive and/or select test cases.
Synonyms: test case design technique, test specification technique, test technique, test design technique

Test tool A software product that supports one or more test activities, such as planning and control, specification, building initial files and data, test execution and test analysis.

Test type A group of test activities based on specific test objectives aimed at specific characteristics of a component or system.

Testability The degree of effectiveness and efficiency with which tests can be designed and executed for a component or system.

Testable requirement A requirements that is stated in terms that permit establishment of test designs (and subsequently test cases) and execution of tests to determine whether the requirement has been met.

Tester A skilled professional who is involved in the testing of a component or system.

Testing The process consisting of all life cycle activities, both static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects.

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

Traceability The degree to which a relationship can be established between two or more work products.

Understandability The capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use. See also: *usability*.

Unit test framework A tool that provides an environment for unit or component testing in which a component can be tested in isolation or with suitable stubs and drivers. It also provides other support for the developer, such as debugging capabilities.

Unreachable code Code that cannot be reached and therefore is impossible to execute.
Synonym: dead code

Usability The degree to which a component or system can be used by specified users to achieve specified goals in a specified context of use.

Usability testing Testing to evaluate the degree to which the system can be used by specified users with effectiveness, efficiency and satisfaction in a specified context of use.

Use case A sequence of transactions in a dialogue between an actor and a component or system with a tangible result, where an actor can be a user or anything that can exchange information with the system.

Use case testing A black-box test technique in which test cases are designed to execute scenarios of use cases.
Synonyms: scenario testing, user scenario testing

User acceptance testing Acceptance testing conducted in a real or simulated operational environment by intended users focusing on their needs, requirements and business processes. See also: *acceptance testing.*

User interface All components of a system that provide information and controls for the user to accomplish specific tasks with the system.

User story A high-level user or business requirement commonly used in Agile software development, typically consisting of one sentence in the everyday or business language capturing what functionality a user needs and the reason behind this, any non-functional criteria and also includes acceptance criteria. See also: *Agile software development, requirement.*

V-model A sequential development life cycle model describing a one-for-one relationship between major phases of software development from business requirements specification to delivery, and corresponding test levels from acceptance testing to component testing.

Validation Confirmation by examination and through provision of objective evidence that the requirements for a specific intended use or application have been fulfilled.

Variable An element of storage in a computer that is accessible by a software program by referring to it by a name.

Verification Confirmation by examination and through provision of objective evidence that specified requirements have been fulfilled.

Walkthrough A type of review in which an author leads members of the review through a work product and the members ask questions and make comments about possible issues. See also: *peer review.*
Synonym: structured walkthrough

White-box test technique A procedure to derive and/or select test cases based on an analysis of the internal structure of a component or system.
Synonyms: structural test technique, structure-based test technique, structure-based technique, white-box technique

White-box testing Testing based on an analysis of the internal structure of the component or system.
Synonyms: clear-box testing, code-based testing, glass-box testing, logic-coverage testing, logic-driven testing, structural testing, structure-based testing

Wideband Delphi An expert-based test estimation technique that aims at making an accurate estimation using the collective wisdom of the team members.

ANSWERS TO SAMPLE EXAM QUESTIONS

This section contains the answers and the learning objectives for the sample questions in each chapter and for the full mock exam in Chapter 7.

If you get any of the questions wrong or if you were not sure about the answer, then the learning objective tells you which part of the Syllabus to go back to in order to help you understand why the correct answer is the right one. The learning objectives are listed at the beginning of each section. For example, if you got Question 3 in Chapter 1 wrong, then go to Chapter 1 and read Learning Objective 1.2.2. Then re-read the section in the chapter which deals with that topic.

CHAPTER 1 FUNDAMENTALS OF TESTING

Question	Answer	Learning objective
1	a	1.2.1
2	b	Keywords
3	c	1.2.2
4	a	1.1.1
5	a	1.3.1
6	c	1.4.4
7	b	1.5.1
8	d	1.4.3

CHAPTER 2 TESTING THROUGHOUT THE SOFTWARE DEVELOPMENT LIFE CYCLE

Question	Answer	Learning objective
1	d	2.1.1
2	d	2.2.1
3	b	Keywords
4	b	2.4.1
5	c	2.3.1
6	d	2.3.3
7	c	2.3.3
8	b	2.3.1
9	a	2.2.1

CHAPTER 3 STATIC TECHNIQUES

Question	Answer	Learning objective
1	d	3.1.1
2	a	3.2.1
3	d	3.1.3
4	a	3.2.3
5	d	3.2.2
6	b	3.2.3
7	a	3.2.5
8	c	3.2.4
9	c	3.2.4

CHAPTER 4 TEST TECHNIQUES

Question	Answer	Learning objective
1	d	4.4.3
2	a	4.2.2
3	c	4.3.3
4	a	4.1.1
5	b	4.1.1
6	c	4.2.3
7	d	4.2.4
8	b	4.2.1
9	b	4.2
10	c	4.3
11	a	4.2.5
12	c	4.3
13	a	4.4
14	c	4.3.1
15	d	4.3
16	b	4.1.1
17	a	4.2.4

CHAPTER 5 TEST MANAGEMENT

Question	Answer	Learning objective
1	b	5.1.1
2	d	5.1.2
3	b	Keywords
4	a	5.2.1
5	c	5.2.5
6	c	5.2.2
7	d	5.2.3
8	a	Keywords
9	c	5.3.1
10	b	5.3.2
11	a	5.6.1
12	c	5.4.1
13	d	5.5.2
14	b	5.5.1
15	a	5.5.2
16	a	5.5.3
17	b	5.2.6
18	d	5.6.1
19	c	5.2.4
20	a	5.2.3

CHAPTER 6 TOOL SUPPORT FOR TESTING

Question	Answer	Learning objective
1	d	6.1.1
2	c	6.1.1
3	b	6.1.2
4	a	6.1.3
5	b	6.2.1
6	d	6.2.2
7	a	6.2.3

CHAPTER 7 MOCK EXAM

Question	Answer	Learning objective
1	b	4.2.1
2	a	1.1.2
3	a	2.1.2
4	d	4.2.5
5	c	5.5.2
6	b	2.3.3
7	c	Chapter 4 Keywords
8	b	5.6.1
9	c	5.1.2
10	c	1.4.1
11	d	3.2.1
12	b	1.4.2
13	a	1.5.2
14	b	5.3.1
15	a	4.3.2
16	a	6.1.1
17	c	4.4.1
18	b	5.2.6
19	a	4.2.1
20	b	Chapters 1/4 Keyword
21	b	3.1.1
22	b	6.2.3
23	a	1.2.3
24	b	2.2.1
25	c	1.2.4
26	c	4.2.2
27	c	4.2.3
28	d	4.4.2
29	b	2.3.1
30	c	5.2.1
31	b	4.1.1
32	c	3.1.3
33	d	1.1.1
34	d	5.4.1
35	c	4.2.4
36	c	5.2.4
37	b	3.2.5
38	a	2.4.2
39	c	3.2.4
40	d	5.1.1

REFERENCES

Key:
In Syllabus
Extra to Syllabus

CHAPTER 1 FUNDAMENTALS OF TESTING

Beizer, B. (1990) *Software Testing Techniques* (2nd edition), Van Nostrand Reinhold: Boston, MA
Black, R. (2004) *Critical Testing Processes*, Addison Wesley: Reading, MA
Black, R. (2009) *Managing the Testing Process* (3rd edition), John Wiley & Sons: New York, NY
Gilb, T. and Graham, D. (1993) *Software Inspection*, Addison Wesley: Reading, MA
ISO/IEC/IEEE 29119-1 (2013) *Software and systems engineering – Software testing – Part 1: Concepts and definitions*
ISO/IEC/IEEE 29119-2 (2013) *Software and systems engineering – Software testing – Part 2: Test processes*
ISO/IEC/IEEE 29119-3 (2013) *Software and systems engineering – Software testing – Part 3: Test documentation*
Jones, C. (2008) *Estimating Software Costs* (3rd edition), McGraw Hill Education: New York, NY
Myers, G., Badgett, T., and Sandler, C. (2012) *The Art of Software Testing* (3rd edition), John Wiley & Sons: New York, NY
Weinberg, G. (2008) *Perfect Software and Other Illusions about Testing*, Dorset House: New York, NY

CHAPTER 2 TESTING THROUGHOUT THE SOFTWARE DEVELOPMENT LIFE CYCLE

Berard, Edward V. (1993) *Essays on Object-oriented Software Engineering* (Volume 1), Prentice Hall: Englewood Cliffs, NJ
Black, R. (2017) *Agile Testing Foundations*, BCS Learning & Development Ltd: Swindon, UK
Boehm, B. (1986) 'A Spiral Model of Software Development and Enhancement', *ACM SIGSOFT Software Engineering Notes*, ACM, 11(4):14–25, August 1986
Crispin, L. and Gregory, J. (2008) *Agile Testing*, Pearson Education: Boston, MA
Gregory, J. and Crispin, L. (2015) *More Agile Testing*, Pearson Education: Boston, MA
ISO/IEC 25010 (2011) *Systems and software engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) System and Software Quality Models*

CHAPTER 3 STATIC TECHNIQUES

Gilb, T. and Graham, D. (1993) *Software Inspection*, Addison-Wesley: London
ISO/IEC 20246 (2017) *Software and system engineering – Work product reviews*
Kramer, A. and Legeard, B. (2016) *Model-Based Testing Essentials: Guide to the ISTQB Certified Model-Based Tester: Foundation Level*, John Wiley & Sons: New York, NY
Mars.jpl.nasa.gov (1999) *Mars climate orbiter failure board releases report, numerous NASA actions underway in response*. [Online] Available at: https://mars.jpl.nasa.gov/msp98/news/mco991110.html [Accessed 02 April 2019]
Sauer, C. (2000) 'The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research', *IEEE Transactions on Software Engineering*, 26(1):1–14
Shull, F., Rus, I. and Basili, V. (2000), 'How Perspective-Based Reading can Improve Requirement Inspections', *IEEE Computer,* 33(7):73–79
van Veenendaal, E. (1999) 'Practical Quality Assurance for Embedded Software', in *Software Quality Professional*, Vol. 1, no. 3, American Society for Quality, June 1999
van Veenendaal, E. (ed.) (2004) *The Testing Practitioner* (Chapters 8–10), UTN Publisher: The Netherlands
van Veenendaal, E. and van der Zwan, M. (2000) 'GQM Based Inspections', in *Proceedings of the 11th European Software Control and Metrics Conference (ESCOM)*, Munich, May 2000
Wiegers, K. (2002) *Peer Reviews in Software*, Pearson Education: Boston, MA

CHAPTER 4 TEST TECHNIQUES

Beizer, B. (1990) *Software Testing Techniques* (2nd edition), Van Nostrand Reinhold: Boston, MA
Black, R. (2007) *Pragmatic Software Testing*, John Wiley & Sons: New York, NY
Broekman, B. and Notenboom, E. (2003) *Testing Embedded Software*, Addison Wesley: London
Copeland, L. (2004) *A Practitioner's Guide to Software Test Design*, Artech House: Norwood, MA
Craig, R. D. and Jaskiel, S. P. (2002) *Systematic Software Testing*, Artech House: Norwood, MA
Gilb, T. (1988) *Principles of Software Engineering Management*, Addison Wesley: Reading, MA
Hetzel, B. (1988) *The Complete Guide to Software Testing* (2nd edition), QED Information Sciences: Wellesley, MA
ISO/IEC/IEEE 29119-4 (2015) *Software and system engineering - Software testing - Part 4: Test techniques*
Jacobson, I. (1992) *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison Wesley: Reading, MA
Jorgensen, P. (2014) *Software Testing: A Craftsman's Approach* (4e), CRC Press: Boca Raton FL
Kaner, C., Bach, J., and Petticord, B. (2002) *Lessons Learned in Software Testing*, John Wiley & Sons: New York, NY
Kaner, C., Padmanabhan, S., and Hoffman, D. (2013) *The Domain Testing Workbook*, Context-driven Press: Orlando, FL
Marick, B. (1994) *The Craft of Software Testing*, Prentice Hall: New York, NY
Myers, G., Badgett, T., and Sandler, C., (2012) *The Art of Software Testing* (3rd edition), John Wiley & Sons: New York, NY
Pol, M., Teunissen, R., and van Veenendaal, E. (2001) *Software Testing: A Guide to the TMap Approach*, Addison Wesley: Harlow, UK
UML 2.5: Omg.org (2017) *About the Unified Modeling Language Specification Version 2.5.1*. [Online] Available at: http://www.omg.org/spec/UML/2.5.1/ [Accessed 02 April 2019]
Whittaker, J. A. (2003) *How to Break Software: A Practical Guide to Testing*, Addison Wesley: Reading, MA

CHAPTER 5 TEST MANAGEMENT

Beizer, B. (1990) *Software Testing Techniques* (2nd edition), Van Nostrand Reinhold: Boston, MA
Black, R. (2004) *Critical Testing Processes*, Addison Wesley: Reading, MA
Black, R. (2009) *Managing the Testing Process* (3rd edition), John Wiley & Sons: New York, NY
Brooks, F. (1995) *The Mythical Man-Month and Other Essays on Software Engineering*, Addison Wesley: New York, NY
Craig, R. D. and Jaskiel, S. P. (2002) *Systematic Software Testing*, Artech House: Norwood, MA
Hetzel, W. (1988) *Complete Guide to Software Testing*, QED: Wellesley, MA
ISO/IEC/IEEE 29119-3 (2013) *Software and systems engineering – Software testing – Part 3: Test documentation*
ISO/IEC 25010 (2011) *Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models*
Kaner, C., Bach, J., and Petticord, B. (2002) *Lessons Learned in Software Testing*, John Wiley & Sons: New York, NY
Pol, M., Teunissen, R., and van Veenendaal, E. (2002) *Software Testing: A Guide to the TMap Approach*, Addison Wesley: Reading, MA
van Veenendaal, E. and Wells, B. (2012) *Test Maturity Model integration TMMi: (Guidelines for Test Process Improvement)*, UTN Publishers: Den Bosch, The Netherlands
Whittaker, J. (2002) *How to Break Software: A Practical Guide to Testing*, Addison Wesley: Reading, MA
Whittaker, J. and Thompson, H.H. (2003) *How to Break Software Security*, Addison Wesley: Reading, MA

CHAPTER 6 TOOL SUPPORT FOR TESTING

Adzic, G. (2011) *Specification by Example*, Manning Publications Co: Shelter Island, NY
Axelrod, A. (2018) *Complete Guide to Test Automation: Techniques, Practices and Patterns for Building and Maintaining Effective Software Projects*: apress: New York, NY
Buwalda, H., Janssen, D., and Pinkster, I. (2001) *Integrated Test Design and Automation*, Addison Wesley: Reading, MA
Fewster, M. and Graham, D. (1999) *Software Test Automation*, Addison Wesley: Reading, MA
Gamba, S. and Graham, D. (2018) *A Journey Through Test Automation Patterns*, amazon
Graham, D. and Fewster, M. (2012) *Experiences of Test Automation*, Pearson Education: Boston, MA

AUTHORS

DOROTHY GRAHAM

Dorothy Graham (Dot) has been involved in software testing for roughly 50 years, since 1970, when her first job (for Bell Labs in the US) was as a programmer in a testing group, and her task was to write two testing tools (test execution and comparison). After emigrating to the UK (with her British husband), she worked for Ferranti Computer Systems, developing software for UK Police forces. She then joined the National Computing Centre as a trainer and courseware developer, and later became an independent consultant.

At that time, software testing was not a respected profession; in fact, in the early 1990s, many thought of testing at best as a necessary evil (if they thought of testing at all!). There were few people who specialized in testing, and it was seen as a second-class activity, and not well thought of. There was a general perception that testing was easy, that anyone could do it, and that you were rather strange if you liked it. It was then that Dot decided to specialize in testing, seeing great scope for improvement in testing activities in industry, not only in imparting fundamental knowledge about testing (basic principles and techniques) but also in improving the view testers had of themselves, and the perceptions of testers in their companies. She developed training courses in testing, and began Grove Consultants, named after her house in Macclesfield, UK. One of her most popular talks at the time was called *Test is a four-letter word*, reflecting the prevailing culture about testing.

It was into this context that the initiative to create a qualification for testers was born. Although not the initiator, Dot was involved from the first meetings and the earliest working groups that developed the first Foundation Syllabus, donating many hours of time to help progress this effort. This work was carried out with support from ISEB (Information Systems Examination Board) of the British Computer Society, and the testing qualification was modelled on ISEB's successful qualifications in Project Management and Information Systems Infrastructure. One of the aims at this time was to give people a common vocabulary to talk about testing, since people seemed to be using many different terms for the same thing.

Grove Consultants (Dorothy Graham and Mark Fewster at that time) gave the first course based on the ISEB Foundation Syllabus in October 1998 and the first Foundation Certificates in Software Testing were awarded. In her 20 years with them, Grove went on to be highly respected for the quality of their training material, particularly for ISTQB courses. Grove continues to licence high-quality ISTQB training material to organizations wishing to provide training without expending a significant effort in course development (www.grove.co.uk).

The success of the Foundation qualification took everyone by surprise. There seemed to be a hunger for a qualification that gave testers more respect, both for themselves and from their employers. It also gave testers a common vocabulary and more confidence in their work. The Foundation qualification had met its main objective of 'removing the bottom layer of ignorance' about software testing.

Work then began on extending the ISEB qualification to a more advanced level (which became the ISEB Practitioner qualification) and also to extending it to other countries, as news of the qualification spread in the international community. Dot was a facilitator at the meeting that formed ISTQB in 2001 in Sollentuna, Sweden. She has not been actively involved in the ISTQB Advanced levels.

Dorothy's other activities over the years include being Programme Chair for the first European testing conference (EuroSTAR 1993); she was Programme Chair again in 2009. During the 1990s, she started and later co-authored *The CAST Report*, a summary of commercial testing tools (in the days before the internet!). In addition to all editions of this book on Foundations of Software Testing, she was co-author of four other books: *Software Inspection* (1993) with Tom Gilb, *Software Test Automation* (1998) and *Experiences of Software Test Automation* (2012), both with Mark Fewster, and *A Journey Through Test Automation Patterns* (2018) with Seretta Gamba.

Her book with Seretta is the story of a team using the test automation patterns wiki, TestAutomationPatterns.org. This wiki, developed by Seretta and Dot, provides solutions that have worked for others for a number of issues and problems in test automation. The issues and patterns are organized into four sections: Process, Management, Design and Execution. The wiki was first published in 2013 and is a popular source of advice about test automation.

During her career, Dot has spoken at numerous testing conferences and events worldwide. She was awarded the second European Excellence Award in Software Testing in 1999 and was awarded the first ISTQB Excellence Award in Software Testing in 2012.

Her main non-testing activities include singing (choirs, madrigal groups and solos) and enjoyable holidays with her husband, Roger. They have two children, Sarah (married to Tim) and James.

Dorothy can be contacted on LinkedIn and www.DorothyGraham.co.uk.

REX BLACK

With over 35 years of software and systems engineering experience, Rex Black is President of RBCS (www.rbcs-us.com), a leader in software, hardware and systems testing. For 25 years, RBCS has delivered consulting, training and expert services for software and hardware testing. Employing the industry's most experienced and recognized consultants, RBCS builds and improves testing groups, trains teams and provides testing experts for hundreds of clients worldwide. Ranging from Fortune 100 companies to start-ups, RBCS clients save time and money through improved product development, decreased tech support calls, improved corporate reputation and more. As the leader of RBCS, Rex is the most prolific author practising in the field of software testing today. More about RBCS is shown after Rex's biography.

Rex's popular first book, *Managing the Testing Process*, has sold over 100,000 copies around the world, including Japanese, Chinese, and Indian releases and is now in its third edition. In addition to *Managing the Testing Process* and *Foundations of Software Testing*, Rex has written 12 other books on testing, including *Advanced Software Testing: Volume I*, *Advanced Software Testing: Volume II, Advanced Software Testing: Volume III, Critical Testing Processes, Pragmatic Software Testing*, *Agile Testing Foundations*, *Mobile Testing*, *Expert Test Manager* and *Fundamentos de Prueba de Software*. These works have also sold tens of thousands of copies, including Hebrew, Indian, Chinese, Japanese and Russian editions. He has written over 30 articles; presented hundreds of papers, workshops, and seminars; and given about 50 keynotes and other speeches at conferences and events around the world.

Rex is the past President of the International Software Testing Qualifications Board and the past President of the American Software Testing Qualifications Board. He remains involved in the ISTQB, having served as Project Manager of the Foundation 2018 Syllabus effort and as Chair of the Agile Working Group.

Rex is married to his college girlfriend, Laurel Becker. They met in 1987 at the University of California, Los Angeles. They have two children, Emma and Charlotte, and three dogs, Kibo, Mmink and Roscoe. They live in Bulverde, Texas and Lake Tahoe, California.

COMPANY PROFILE: RBCS, INC.

Rex Black Consulting Services (RBCS) is a premier international testing consultancy specializing in consulting, training and expert services. Led by one of the most recognized and published industry leaders, Rex Black, RBCS is an established company you can trust to show you results, with the track record to prove it.

Since 1994, RBCS has been both a pioneer and leader in quality hardware and software testing. Through its training, consulting and expert services, RBCS has helped hundreds of companies improve their test practices and achieve quality results. RBCS is based in the US, and works with partners around the world.

RBCS utilizes deep industry experience to solve testing problems and improve testing processes. It helps clients reduce risk and save money through higher-quality products and services. The consultancy's goal is to help clients avoid the costs associated with poor product quality (such as tech support calls, loss of business, customer dissatisfaction, lawsuits and reputation damage) by helping them understand and solve testing and performance issues and build better testing teams. Whether it's customizing a program to fit your company's needs, or providing the hands-on experience and resources that will allow your testing team to grow professionally, RBCS helps companies produce better products and increase ROI. RBCS pours its resources into ensuring our clients become successful.

Employing the industry's most experienced and recognized consultants, RBCS builds and improves testing groups, trains teams and provides testing experts for hundreds of clients worldwide. RBCS's principals include published, international experts in the area of hardware and software testing. Their skilled test managers and engineers all hold International Software Testing Qualifications Board (ISTQB) certifications and are highly trained.

RBCS believes in the value of real-life experience and leadership which is why the RBCS team is always striving to improve and perfect the testing process. Every trainer and consultant is ISTQB certified and encouraged to write articles and books, share leading ideas through presentations, and continuously research new ideas. In addition, as past President of the ISTQB and the American Software Testing Qualifications Board (ASTQB), Rex Black has more than 35 years of software and systems engineering experience and strongly leads the company with the most recent, proven testing methodologies and strategies.

RBCS' client-centred approach helps companies deliver better quality products and reduce costs by improving their test practices. The difference? RBCS's recognized industry experts started their careers in our customers' shoes; they have the real-world knowledge required to deliver the best services with proven practices.

From Fortune 100 to small and mid-size organizations, RBCS works with a variety of companies, in industries such as technology, finance, communications, retail, government and education.

RBCS offers customized consulting services that will not only help you solve your testing challenges, but provide you with the tools and framework for managing a successful testing organization. Our consulting methodology is based on successful, peer-reviewed publications with a customized approach that focuses on the individual needs of each client. Plus we have a long list of happy, referenceable customers around the globe that have benefited from RBCS expertise.

RBCS, Inc.
31,520 Beck Road
Bulverde, TX 78163
USA
www.rbcs-us.com
email: info@rbcs-us.com
Web: www.rbcs-us.com
LinkedIn: www.linkedin.com/in/rex-black
YouTube: www.youtube.com/user/RBCSINC
Facebook: www.fb.me/TestingImprovedbyRBCS
Twitter: @RBCS

ERIK VAN VEENENDAAL

Erik van Veenendaal is an internationally recognized testing expert, author of a number of books and has published a large number of papers within the profession. He is currently working as an independent consultant and as the Chief Executive Officer (CEO) of the TMMi Foundation.

Dr Erik van Veenendaal CISA graduated at the University of Tilburg in Business Economics. He has been working as a practitioner and manager in the IT industry since 1987. After a career in software development, he moved to the area of software quality, where he specializes in software testing.

As a test manager and test consultant he has been involved in a great number and variety of projects. He has implemented structured testing and reviews and inspections, and as a consultant has contributed to many test process improvement projects. He worked for Sogeti as manager of operations and was one the core developers of the TMap testing methodology. He is the author of numerous papers and a number of books on testing and software, including the best sellers *The Testing Practitioner, The Little TMMi, Foundations of Software Testing* and *Testing according to TMap.*

Erik van Veenendaal founded Improve Quality Services BV, a company that provides consultancy and training services in the areas of testing, requirements engineering and quality management. He has been the company director for over 12 years. Within this period Improve Quality Services became a leading testing company in

The Netherlands focused on innovative and high-quality testing services. Customers were especially to be found in the areas of embedded software (e.g., Philips, Océ en Assembléon) and in the finance domain (e.g., Rabobank, ING and Triodos Bank). Improve Quality Services was market leader for test training in The Netherlands both in terms of quantity and quality.

Nowadays Erik is living in Bonaire (Caribbean Netherlands) and is occupied as an independent consultant doing consultancy and training in the areas of testing (especially based on ISTQB Syllabi) and requirements engineering. He also publishes and delivers keynote presentations on a regular basis. In addition, he is actively involved in the TMMi Foundation, International Software Testing Qualifications Board and the IREB requirements engineering organization.

Erik, being one the initiators to found the TESTNET organization, is now an honorary member of TESTNET (the Dutch Special Interest Group in Software Testing). Erik was the first person to receive the ISEB Practitioner certificate with distinction and is also a Certified Information Systems Auditor (CISA). Erik has also been a senior lecturer at the Eindhoven University of Technology, Faculty of Technology Management for almost ten years.

Since its foundation in 2002, Erik has been strongly involved in the International Software Testing Qualifications Board (ISTQB). From 2005 till 2009 he was the vice-president of the ISTQB organization and he is the founder of the local Belgium and The Netherlands board; the Belgium Netherlands Testing Qualifications Board (BNTQB). For many years, he was the editor of the ISTQB *Standard Glossary of Terms used in Software Testing* and chair for the ISTQB Expert level working party. Today, he is the president of the Curaçao Testing Qualifications Board (CTQB). For his major contribution to the field of testing, Erik received the European Testing Excellence Award in December 2007 and the ISTQB International Testing Excellence Award in October 2015.

Erik can be contacted via email at erik@erikvanveenendaal.nl and through his website www.erikvanveenendaal.nl.

INDEX