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 certain types of tool: test execution tools, performance testing tools, static analysis tools and test management tools.

6.1 TYPES OF TEST TOOL

1 **Classify different types of test tools according to the test process activi
ties. (K2)**

2 **Recognize tools that may help developers in their testing. (Kl)**

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 not mention any 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.

6.1.1 Test tool classification

The tools are grouped 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 (sometimes called a 'point solution'), but many of the commercial 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, incident management, and requirements management and traceability; another tool may provide both coverage measurement and test design support.

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's a good idea to use computers to do things that computers are really good at and that people are not very good at. So 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.

A tool that measures some aspect of software may have unexpected side-effects on that software. For example, a tool that measures timings for non-functional (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, i.e. storing the time at those two points, could actually make the whole transaction take slightly longer than it would do if the tool wasn't 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, e.g. 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 re-appears when the debugger is turned off (thereby making it much more difficult to find). These are sometimes known as 'Heizenbugs' (after Heizenberg's uncertainty principle).

In the descriptions of the tools below, we will indicate the tools which are more likely to be used by developers during component testing and component integration testing. 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 and what they do; you do not need a detailed understanding of them (or know how to use them).

6.1.2 Tool support for management of testing and tests

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

The features provided by **test management tools** include those listed below. Some tools will provide all of these features; others may provide one or more of the features, however such tools would still be classified as test management tools.

Features or characteristics of test management tools include support for:

- management of tests (e.g. keeping track of the associated data for a given set of tests, knowing which tests need to run in a common environment, number of tests planned, written, run, passed or failed);
- scheduling of tests to be executed (manually or by a test execution tool);
- management of testing activities (time spent in test design, test execution, whether we are on schedule or on budget);
- interfaces to other tools, such as:
  - test execution tools (test running tools);
  - incident management tools;
  - requirement management tools;
  - configuration management tools;
- traceability of tests, test results and defects to requirements or other sources;
- logging test results (note that the test management tool does not run tests,

but could summarize results from test execution tools that the test management tool interfaces with);

- preparing progress reports based on metrics (quantitative analysis), such as:
  - tests run and tests passed;
  - incidents raised, defects fixed and outstanding.

This information 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.

Some requirements management tools are able to find defects in the requirements, for example by checking for ambiguous or forbidden words, such as 'might', 'and/or', 'as needed' or '(to be decided)'.

Features or characteristics of requirements management tools include support for:

- storing requirement statements;
- storing information about requirement attributes;
- checking consistency of requirements;
- identifying undefined, missing or 'to be defined later' requirements;
- prioritizing requirements for testing purposes;
- traceability of requirements to tests and tests to requirements, functions or features;
- traceability through levels of requirements;
- interfacing to test management tools;
- coverage of requirements by a set of tests (sometimes).

Incident management tools

This type of tool is also known as a defect-tracking tool, a defect-management tool, a bug-tracking tool or a bug-management tool. However, **'incident management tool'** is probably a better name for it because not all of the things tracked are actually defects or bugs; incidents may also be perceived problems, anomalies (that aren't necessarily defects) or enhancement requests. 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 (e.g. a developer) begins to investigate the failure.

Incident 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. Incident management tools make it much easier to keep track of the incidents over time.

Features or characteristics of incident management tools include support for:

- storing information about the attributes of incidents (e.g. severity);
- storing attachments (e.g. a screen shot);
- prioritizing incidents;
- assigning actions to people (fix, confirmation test, etc.);
- status (e.g. open, rejected, duplicate, deferred, ready for confirmation test, closed);
- reporting of statistics/metrics about incidents (e.g. average time open, number of incidents with each status, total number raised, open or closed).

Incident management tool functionality may be included in commercial test management tools.

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 weren't 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.

Testware needs to be under configuration management and the same tool may be able to 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 earlier example shows.

Features or characteristics of configuration management tools include support for:

- storing information about versions and builds of the software and testware;
- traceability between software and testware and different versions or variants;
- keeping track of which versions belong with which configurations (e.g. oper ating systems, libraries, browsers);
- build and release management;
- baselining (e.g. all the configuration items that make up a specific release);
- access control (checking in and out).

6.1.3 Tool support for static testing

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

Review process support tools

The value of different types of review was discussed in Chapter 3. For a very informal review, where one person looks at another's document 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** which 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 document too quickly, i.e. that the checking rate (number of pages checked per hour) was close to that recommended for that review cycle. A review process support tool could automatically calculate the checking rate and flag exceptions. The review process support tools can normally be tailored for the particular review process or type of review being done.

Features or characteristics of review process support tools include support for:

- a common reference for the review process or processes to use in different situations;
- storing and sorting review comments;
- communicating comments to relevant people;
- coordinating online reviews;
- keeping track of comments, including defects found, and providing statisti cal information about them;
- providing traceability between comments, documents reviewed and related documents;
- a repository for rules, procedures and checklists to be used in reviews, as well as entry and exit criteria;
- monitoring the review status (passed, passed with corrections, requires re-review);
- collecting metrics and reporting on key factors.

Static analysis tools (D)

The '(D)' after this (and other types of tool) indicates that these tools are more likely to be used by developers. **Static analysis** by tools was discussed in Chapter 3. In this section we give a summary of what the tools do.

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 is worth checking what is available from existing compilers or development environments before looking at purchasing 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 (for example, 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. See Section 6.2.3 for special considerations when introducing static analysis tools into an organization.

Features or characteristics of static analysis tools include support to:

- calculate metrics such as cyclomatic complexity or nesting levels (which can help to identify where more testing may be needed due to increased risk);
- enforce coding standards;
- analyze structures and dependencies;
- aid in code understanding;
- identify anomalies or defects in the code (as described in Chapter 3).

Modeling tools (D)

Modeling tools help to validate models of the system or software. For example a tool can check consistency of data objects in a database and can find inconsistencies and defects. These may be difficult to pick up in testing - you may have tested with one data item and not realize that in another part of the database there is conflicting information related to that item. Modeling tools can also check state models or object models.

Modeling tools are typically used by developers and can help in the design of the software.

One strong advantage of both modeling tools and static analysis tools is that they can be used before dynamic tests can be run. This enables any defects that these tools can find to be identified as early as possible, when it is easier and cheaper to fix them. There are also fewer defects left to propagate into later stages, so development can be speeded up and there is less rework. (Of course this is difficult to show, since those defects aren't there now!)

Note that 'model-based testing tools' are actually tools that generate test inputs or test cases from stored information about a particular model (e.g. a state diagram), so are classified as test design tools (see Section 6.1.4).

Features or characteristics of modeling tools include support for:

- identifying inconsistencies and defects within the model;
- helping to identify and prioritize areas of the model for testing;
- predicting system response and behavior under various situations, such as level of load;
- helping to understand system functions and identify test conditions using a modeling language such as UML.

6.1.4 Tool support for test specification

The tools described in this section support the testing activities described in Chapter 4.

Test design tools

Test design tools help to construct test cases, or at least test inputs (which is 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).

For example, if the requirements are kept in a requirements management or test management tool, or in a Computer Aided Software Engineering (CASE) tool used by developers, then it is possible to identify the input fields, including the range of valid values. This range information can be used to identify boundary values and equivalence partitions. If the valid range is stored, the tool can distinguish between values that should be accepted and those that should generate an error message. If the error messages are stored, then the expected result can be checked in detail. If the expected result of the input of a valid value is known, then that expected result can also be included in the test case constructed by the test design tool.

Another type of test design tool is one that helps to select combinations of possible factors to be used in testing, to ensure that all pairs of combinations of operating system and browser are tested, for example. Some of these tools may use orthogonal arrays. See [Copeland, 2003] for a description of these combination techniques.

Note that the test design tool may have only a partial oracle - that is, it may know which input values are to be accepted and rejected, but it may not know the exact error message or resulting calculation for the expected result of the test. Thus the test design tool can help us to get started with test design and will identify all of the fields, but it will not do the whole job of test design for us - there will be more verification that may need to be done.

Another type of test design tool is sometimes called a 'screen scraper', a structured template or a test frame. The tool looks at a window of the graphical user interface and identifies all of the buttons, lists and input fields, and can set up a test for each thing that it finds. This means that every button will be clicked for example and every list box will be selected. This is a good start for a thorough set of tests and it can quickly and easily identify non-working buttons. However, unless the tool has access to an oracle, it may not know what should actually happen as a result of the button click.

Yet another type of test design tool may be bundled with a coverage tool. If a coverage tool has identified which branches have been covered by a set of existing tests for example, it can also identify the path that needs to be taken in order to cover the untested branches. By identifying which of the previous decision outcomes need to be True or False, the tool can calculate an input value that will cause execution to take a particular path in order to increase coverage. Here the test is being designed from the code itself. In this case the presence of an oracle is less likely, so it may only be the test inputs that are constructed by the test design tool.

Features or characteristics of test design tools include support for:

- generating test input values from:
  - requirements;
  - design models (state, data or object);
  - code;
  - graphical user interfaces;
  - test conditions;
- generating expected results, if an oracle is available to the tool.

The benefit of this type of tool is that it can easily and quickly identify the tests (or test inputs) that will exercise all of elements, e.g. input fields, buttons, branches. This helps the testing to be more thorough (if that is an objective of the test!)

Then we may have the problem of having too many tests and need 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). Using a combination technique such as orthogonal arrays can also help.

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.

Test data preparation tools enable data to be selected from an existing database or created, generated, manipulated and edited for use in tests. The most sophisticated tools can deal with a range of files and database formats.

Features or characteristics of test data preparation tools include support to:

- extract selected data records from files or databases;
- 'massage' data records to make them anonymous or not able to be identified with real people (for data protection);
- enable records to be sorted or arranged in a different order;
- generate new records populated with pseudo-random data, or data set up according to some guidelines, e.g. an operational profile;
- construct a large number of similar records from a template, to give a large set of records for volume tests, for example.

6.1.5 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'. Most 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 program, and playing it back. However, the tests are not something which is played back just for someone to watch the tests interact with the system, which may react slightly differently when the tests are repeated. Hence captured tests are not suitable if you want to achieve long-term success with a test execution tool, as is described in Section 6.2.3.

Test execution tools use a scripting language to drive the tool. The scripting language is actually a programming language. So any tester who wishes to use a test execution tool directly will need to use programming skills to create and modify the scripts. The advantage of programmable scripting is that tests can repeat actions (in loops) for different data values (i.e. test inputs), they can take different routes depending on the outcome of a test (e.g. if a test fails, go to a different set of tests) and they can be called from other scripts giving some structure to the set of tests.

When people first encounter a test execution tool, they tend to use it to 'capture/playback', which sounds really good when you first hear about it. The theory is that while you are running your manual tests, you simply turn on the 'capture', like a video recorder for a television program. However, the theory breaks down when you try to replay the captured tests - this approach does not scale up for large numbers of tests. 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.
- 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 sub sequent requests that contain the previously captured order number.
- The test input information is 'hard-coded', i.e. it is embedded in the individ ual 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 scripting is an advance over captured scripts but keyword-driven scripts give significantly more benefits. [Fewster and Graham, 1999], [Buwalda *et al.*, 2001]. [Mosley and Posey, 2002] describe 'control synchronized data-driven testing'. See also 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. For example, a test execution tool can help to identify the input fields which will form test inputs and may construct a table which is the first step towards data-driven scripting.

Although they are commonly referred to as testing tools, they are actually best used for regression testing (so they could be referred to as 'regression testing tools' rather than 'testing tools'). A test execution tool most often runs tests that have already been run before. One of the most significant benefits of using this type of tool is that whenever an existing system is changed (e.g. for a defect fix or an enhancement), *all* of the tests that were run earlier could potentially be run again, to make sure that the changes have not disturbed the existing system by introducing or revealing a defect.

Features or characteristics of test execution tools include support for:

- capturing (recording) test inputs while tests are executed manually;
- storing an expected result in the form of a screen or object to compare to, the next time the test is run;
- executing tests from stored scripts and optionally data files accessed by the script (if data-driven or keyword-driven scripting is used);
- dynamic comparison (while the test is running) of screens, elements, links, controls, objects and values;
- ability to initiate post-execution comparison;
- logging results of tests run (pass/fail, differences between expected and actual results);
- masking or filtering of subsets of actual and expected results, for example excluding the screen-displayed current date and time which is not of interest to a particular test;
- measuring timings for tests;
- synchronizing inputs with the application under test, e.g. wait until the appli cation is ready to accept the next input, or insert a fixed delay to represent human interaction speed;
- sending summary results to a test management tool.

Test harness/unit test framework tools (D)

These two types of tool are grouped together because they are variants of the type of support needed by developers when testing individual components or units of software. A **test harness** provides **stubs** and **drivers,** which are small programs that interact with the software under test (e.g. for testing middle-ware and embedded software). See Chapter 2 for more detail of how these are used in integration testing. Some **unit test framework tools** provide support for object-oriented software, others for other development paradigms. Unit test frameworks can be used in agile development to automate tests in parallel with development. Both types of tool enable the developer to test, identify and localize any defects. The framework or the stubs and drivers supply any information needed by the software being tested (e.g. an input that would have come from a user) and also receive any information sent by the software (e.g. a value to be displayed on a screen). Stubs may also be referred to as 'mock objects'.

Test harnesses or drivers may be developed in-house for particular systems. Advice on designing test drivers can be found in [Hoffman and Strooper, 1995].

There are a large number of 'xUnit' tools for different programming languages, e.g. JUnit for Java, NUnit for .Net applications, etc. There are both commercial tools and also open-source (i.e. free) 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, i.e. for component or component integration testing, rather than for system or acceptance testing.

Features or characteristics of test harnesses and unit test framework tools include support for:

- supplying inputs to the software being tested;
- receiving outputs generated by the software being tested;
- executing a set of tests within the framework or using the test harness;
- recording the pass/fail results of each test (framework tools);
- storing tests (framework tools);
- support for debugging (framework tools);
- coverage measurement at code level (framework tools).

Test comparators

Is it really a test if you put some inputs into some software, but never look to see whether the software produces the correct result? The essence of testing is to check whether the software produces the correct result, and to do that, we must compare what the software produces to what it should produce. A **test comparator** helps to automate aspects of that comparison.

There are two ways in which actual results of a test can be compared to the expected results for the test. Dynamic comparison is where the comparison is done dynamically, i.e. while the test is executing. The other way is post-execution comparison, where the comparison is performed after the test has finished executing and the software under test is no longer running.

Test execution tools include the capability to perform dynamic comparison while the tool is executing a test. This type of comparison is good for comparing the wording of an error message that pops up on a screen with the correct wording for that error message. Dynamic comparison is useful when an actual result does not match the expected result in the middle of a test - the tool can be programmed to take some recovery action at this point or go to a different set of tests.

Post-execution comparison is usually best done by a separate tool (i.e. not the test execution tool). This is the type of tool that we mean by a test comparator or test comparison tool and is typically a 'stand-alone' tool. Operating systems normally have file comparison tools available which can be used for post-execution comparison and often a comparison tool will be developed in-house for comparing a particular type of file or test result.

Post-execution comparison is best for comparing a large volume of data, for example comparing the contents of an entire file with the expected contents of that file, or comparing a large set of records from a database with the expected content of those records. For example, comparing the result of a batch run (e.g. overnight processing of the day's online transactions) is probably impossible to do without tool support.

Whether a comparison is dynamic or post-execution, the test comparator needs to know what the correct result is. This may be stored as part of the test case itself or it may be computed using a test oracle. See Chapter 4 for information about test oracles.

Features or characteristics of test comparators include support for:

- dynamic comparison of transient events that occur during test execution;
- post-execution comparison of stored data, e.g. in files or databases;
- masking or filtering of subsets of actual and expected results.

Coverage measurement tools (D)

How thoroughly have you tested? **Coverage tools** can help answer this question.

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 (e.g. 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. Although coverage can be measured at system or acceptance testing levels, e.g. where the coverage item may be a requirement statement, there aren't many (if any) commercial tools at this level; there is more tool support at component testing level or to some extent at component integration level.

The process of identifying the coverage items at component test level is called 'instrumenting the code', as described in Chapter 4. A suite of tests is then run through the instrumented code, either automatically using a test execution tool or manually. The coverage tool then 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 (i.e. not yet tested). Additional tests can then be run to increase coverage (the tool reports accumulated coverage of all the tests run so far).

The more sophisticated coverage tools can provide support to help identify the test inputs that will exercise the paths that include as-yet unexercised coverage items (or link to a test design tool to identify the unexercised items). For example, if not all decision outcomes have been exercised, the coverage tool can identify the particular decision outcome (e.g. a False exit from an IF statement) that no test has taken so far, and may then also be able to calculate the test input required to force execution to take that decision outcome.

Features or characteristics of coverage measurement tools include support for:

- identifying coverage items (instrumenting the code);
- calculating the percentage of coverage items that were exercised by a suite of tests;'
- reporting coverage items that have not been exercised as yet;
- identifying test inputs to exercise as yet uncovered items (test design tool functionality);
- generating stubs and drivers (if part of a unit test framework).

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

Security tools

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.

Features or characteristics of security testing tools include support for:

- 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;
- identifying weaknesses in password files and passwords;
- security checks during operation, e.g. for checking integrity of files, and intrusion detection, e.g. checking results of test attacks.

6.1.6 Tool support for performance and monitoring

The tools described in this section support testing that can be carried out on a system when it is operational, i.e. while it is running. This can be during testing or could be after a system is released into live operation.

Dynamic analysis tools (D)

Dynamic analysis tools are 'dynamic' 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 (e.g. 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 isn't 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.

Features or characteristics of dynamic analysis tools include support for:

- detecting memory leaks;
- identifying pointer arithmetic errors such as null pointers;
- identifying time dependencies.

These tools would typically be used by developers in component testing and component integration testing, e.g. when testing middleware, when testing security or when looking for robustness defects.

Another form of dynamic analysis for websites is to check whether each link does actually link to something else (this type of tool may be called a 'web spider'). The tool doesn't know if you have linked to the correct page, but at least it can find dead links, which may be helpful.

Performance-testing, load-testing and stress-testing tools **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, e.g. 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).

In order to assess performance, the tool needs to generate some kind of activity on the system, and this can be done in different ways. At a very simple level the same transaction could be repeated many times, but this is not realistic. There are many levels of realism that could be set, depending on the tool, such as 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.

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.

Features or characteristics of performance-testing tools include support for:

- generating a load on the system to be tested;
- measuring the timing of specific transactions as the load on the system varies;
- measuring average response times;
- producing graphs or charts of responses over time.

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.

Features or characteristics of monitoring tools include support for:

- identifying problems and sending an alert message to the administrator (e.g. network administrator);
- logging real-time and historical information;
- finding optimal settings;
- monitoring the number of users on a network;
- monitoring network traffic (either in real time or covering a given length of time of operation with the analysis performed afterwards).

6.1.7 Tool support for specific application areas (Kl)

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.

Commercial tool sets may be bundled for specific application areas such as web-based or embedded systems.

6.1.8 Tool support using other tools

The tools described in this chapter are not the only tools that a tester can make use of. You may not normally think of a word processor or a spreadsheet as a testing tool, but they are often used to store test designs, test scripts or test data. Testers may also use SQL to set up and query databases containing test data. Tools used by developers when **debugging,** to help localize defects and check their fixes, are also testing tools.

Developers use **debugging tools** when identifying and fixing defects. The debugging tools enable them to run individual and localized tests to ensure that they have correctly identified the cause of a defect and to confirm that their change to the code will indeed fix the defect.

It is a good idea to look at any type of tool available to you for ways it could be used to help support any of the testing activities. For example, testers can use Perl scripts to help compare test results.

6.2 EFFECTIVE USE OF TOOLS: POTENTIAL BENEFITS AND RISKS

1 **Summarize the potential benefits and risks of test automation and tool**
support for testing. (K2)
2 **Recognize that test execution tools can have different scripting tech**
niques, including data-driven and keyword-driven. (Kl)

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

Advice on introducing tools into an organization can be found in web articles, magazines and books such as [Dustin *et al.,* 1999], [Siteur, 2005] and [Fewster and Graham, 1999].

6.2.1 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 of repetitive work;
- greater consistency and repeatability;
- objective assessment;
- ease of access to information about tests or testing.

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 creating a specific test database (which can be done by a test data preparation tool).

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 or possibly a requirements management tool).

If a person calculates a value from the software or incident 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 cyclomatic complexity or nesting levels of a component (which can be done by a static analysis tool), coverage (coverage measurement tool), system behavior (monitoring tools) and incident statistics (test management tool).

Having lots of data doesn't 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 or test management tool), incident rates (incident 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 sales information available for the type of tool. It is worth investigating a number of different tools to get a general view of the benefits.

6.2.2 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:

- unrealistic expectations for the tool;
- underestimating the time, cost and effort for the initial introduction of a tool;
- underestimating the time and effort needed to achieve significant and con tinuing benefits from the tool;
- underestimating the effort required to maintain the test assets generated by the tool;
- over-reliance on the tool.

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.

Introducing something new into an organization is seldom straightforward. Having purchased a tool, you will want to move from opening the box 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.

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 is possible. Fortunately there are some short-cuts (e.g. reading books and articles about other people's experiences and learning from them). See also Section 6.3 for more detail on introducing a tool into an organization.

Insufficient planning for maintenance of the assets that the tool produces 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.

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. For example, 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. A test that takes a very long time to automate and will not be run very often is better done manually.

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.

6.2.3 Special considerations for some types of tools

Test execution tools

A 'capture/playback tool' is a misleading term, although it is in common use. Capture/playback is one mode of using a test execution tool and probably the worst way to use it!

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 becomes 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.

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 man agement);
- **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 number of control scripts that implement the tests described in the file).

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 doesn't 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 (e.g. a file that already exists) will not be interpreted correctly by the tool.

However, there are some times when capturing test inputs (i.e. 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 log 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.

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

Data-driven scripts allow the data, i.e. the test inputs and expected outcomes, to be stored separately from the script. This has the advantage that a tester who doesn't 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 scripts include not just data but also keywords in the data file or spreadsheet. This enables a tester (who is not a script programmer) to devise a great variety of tests (not just the test 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 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.

Of course, 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 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 don't 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.

More information on data-driven and keyword-driven scripting can be found in [Fewster and Graham, 1999].

Performance testing tools

Performance testing is developing into a specialized discipline of its own. With functional testing, the types of defect that we are looking for are functionality - does the system or component produce the correct result for given inputs? In performance testing, we are not normally concerned so much with functional correctness, but with non-functional quality characteristics. When using a performance testing tool we are looking at the transaction throughput, the degree of accuracy of a given computation, the computer resources being used for a given level of transactions, the time taken for certain transactions or the number of users that can use the system at once.

In order to get the best from a performance-testing tool, it is important to understand what the tool can and cannot do for you. Although this is true for other types of tool as well, there are particular issues with performance-testing tools, including:

- the design of the load to be generated by the tool (e.g. random input or according to user profiles);
- timing aspects (e.g. inserting delays to make simulated user input more realistic);
- the length of the test and what to do if a test stops prematurely;
- narrowing down the location of a bottleneck;
- exactly what aspects to measure (e.g. user interaction level or server level);
- how to present the information gathered.

Static analysis tools

Static analysis tools are very useful to developers, as they can identify potential problems in code before the code is executed and they can also help to check that the code is written to coding standards.

When a static analysis tool is first introduced, there can be some problems. For example, if the tool checks the current coding standard against code written several years ago, there may be a number of things found in the old code that fail to meet the new coding standard now in place. If the old code has been working well for years, it is probably not a good idea to change it just to satisfy the new coding standard (unless changes are necessary for some other reason). There is a risk that the changes to meet the new standard may have inadvertent side-effects which may not be picked up by regression testing.

Static analysis tools can generate a large number of messages, for example by finding the same thing every few lines. This can be rather annoying, especially if the things found are not considered important now, for example warnings rather than potential defects.

The aim of the static analysis tool is to produce code that will be easier to maintain in the future, so it would be a good idea to implement higher standards on new code that is still being tested, before it is released into use, but to allow older code to be less stringently checked. There is still a risk that the changes to conform to the new standard will introduce an unexpected side-effect, but there is a much greater likelihood that it will be found in testing and there is time to fix it before the system is released.

A filter on the output of. the static analysis tool could eliminate some of the less important messages and make the more important messages more likely to be noticed and fixed.

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.

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 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 vendor-specific processes. 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.

6.3 INTRODUCING A TOOL INTO AN ORGANIZATION

1 **State the main principles of introducing a tool into an organization.
(Kl)**
2 **State the goals of a proof-of-concept or piloting phase for tool evalua
tion. (Kl)**
3 **Recognize that factors other than simply acquiring a tool are required
for good tool support. (Kl)**

6.3.1 Main principles

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 and 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 (e.g. readiness for change);
- identification of the areas within the organization where tool support will help to improve testing processes;
- evaluation of tools against clear requirements and objective criteria;
- proof-of-concept to see whether the product works as desired and meets the requirements and objectives defined for it;
- evaluation of the vendor (training, support and other commercial aspects) or open-source network of support;
- identifying and planning internal implementation (including coaching and mentoring for those new to the use of the tool).

6.3.2 Pilot project

One of the ways to do a proof-of-concept is to have a pilot project as the first thing done with a new tool. This 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 concept *is* proven, i.e. that 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 settings for a static analysis tool, 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 new tool are:

- to learn more about the tool (more detail, more depth);
- to see how the tool would fit with existing processes or documentation, how those would need to change to work well with the tool and how to use the tool to streamline existing processes;
- to decide on standard ways of using the tool that will work for all potential users (e.g. naming conventions, creation of libraries, defining modularity, where different elements will be stored, how they and the tool itself will be maintained);
- to evaluate the pilot project against its objectives (have the benefits been achieved at reasonable cost?).

6.3.3 Success factors

Success is not guaranteed or automatic when implementing a testing tool, but many organizations have succeeded. Here are some of the factors that have contributed to success:

- incremental roll-out (after the pilot) to the rest of the organization;
- 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 training, coaching and mentoring of new users;
- defining and communicating guidelines for the use of the tool, based on what was learned in the pilot;
- implementing a continuous improvement mechanism 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.

More information and advice about selecting and implementing tools can be found in [Fewster and Graham, 1999] and [Dustin *et al.*, 1999].

CHAPTER REVIEW

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

From Section 6.1, 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 tool;
- requirements management tool;
- incident management tool;
- configuration management tool.

Tools that support static testing:
- review process support tool;
- static analysis tool (D);
- modeling tool (D).

Tools that support test specification:
- test design tool;
- test data preparation tool.

Tools that support test execution and logging:
- test execution tool;
- test harness and unit test framework tool (D);
- test comparator;
- coverage measurement tool (D);
- security tool.

Tools that support performance and monitoring:
- dynamic analysis tool;
- performance-testing, load-testing and stress-testing tool;
- monitoring tool.

In addition to the tools already listed, you should know the glossary terms **debugging tool, driver, probe effect** and **stub.**

From Section 6.2, you should be able to summarize the potential benefits and potential risks of tool support for testing in general. You should recognize that some tools have special considerations, including test execution tools, performance-testing tools, static analysis tools and test management tools. You should know the glossary terms **data-driven testing, keyword-driven testing** and **scripting language** and recognize these as associated with test execution tools.

From Section 6.3, you should be able to state the main principles of introducing a tool into an organization (e.g. assessing organizational maturity, clear requirements and objective criteria, proof-of-concept, vendor evaluation, coaching and mentoring). You should be able to state the goals of a proof-of-concept or piloting phase for tool evaluation (e.g. learn about the tool, assess fit with current practices, decide on standards, assess benefits). You should recognize that simply acquiring a tool is not the only factor in achieving good tool support; there are many other factors that are important for success (e.g. incremental roll-out, adapting processes, training and coaching, defining usage guidelines, learning lessons and monitoring benefits). There are no specific definitions for this section.

SAMPLE EXAM QUESTIONS

Question 1 Which tools help to support static testing?

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

Question 2 Which test activities are supported by test harness or unit test framework tools?

a. Test management and control.
b. Test specification and design.
c. Test execution and logging.
d. Performance and monitoring.

Question 3 What are the potential benefits from using tools in general to support testing?

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 What is a potential risk in using tools to support testing?

a. Unrealistic expectations, expecting the tool to do too much.
b. Insufficient reliance on the tool, i.e. still doing manual testing when a test execution tool has been purchased.
c. The tool may find defects that aren't there.
d. The tool will repeat exactly the same thing it did the previous time.

Question 5 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 6 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 7 Which of the following is a goal for a proof-of-concept or pilot phase for tool evaluation?

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.

MOCK EXAM

On the real exam, you will have 60 minutes 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.

Question 1 What is a key characteristic of specification-based testing techniques?

a. Tests are derived from information about how the software is constructed.

b. Tests are derived from models (formal or informal) that specify the problem to be solved by the software or its components.

c. Tests are derived based on the skills and experience of the tester.

d. Tests are derived from the extent of the coverage of structural elements of the system or components.

Question 2 An exhaustive test suite would include:

a. All combinations of input values and preconditions.

b. All combinations of input values and output values.

c. All pairs of input value and preconditions.

d. All states and state transitions.

Question 3 Which statement about testing is true?

a. Testing is started as early as possible in the life cycle.

b. Testing is started after the code is written so that we have a system with which to work.

c. Testing is most economically done at the end of the life cycle.

d. Testing can only be done by an independent test team.

Question 4 For a test procedure that is checking modifications of customers on a database, which two steps below would be the lowest priority if we didn't have time to execute all of the steps?

1 Open database and confirm existing customer

2 Change customer's marital status from single to married

3 Change customer's street name from Parks Road to Park Road

4 Change customer's credit limit from 500 to 750

5 Replace customer's first name with exactly the same first name

6 Close the customer record and close the database

a. Tests 1 and 4

b. Tests 2 and 3

c. Tests 5 and 6

d. Tests 3 and 5

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

I An incorrect calculation of fees might shortchange 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. Ill 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 Which of the following could be used to assess the coverage achieved for structure-based (white-box) test techniques?

V Decision outcomes exercised

W Partitions exercised

X Boundaries exercised

Y Conditions or multiple conditions exercised

Z Statements exercised

a. V,WorY

b. WXorY

c. V,YorZ

d. W,XorZ

Question 8 Review the following portion of an incident report.

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 incident report is missing from this incident 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 following are benefits and which are risks of using tools to support testing?

1 Over-reliance on the tool

2 Greater consistency and repeatability

3 Objective assessment

4 Unrealistic expectations

5 Underestimating the effort required to maintain the test assets generated by the tool

6 Ease of access to information about tests or testing

7 Repetitive work is reduced

a. Benefits: 3,4,6 and 7. Risks: 1,2 and 5

b. Benefits: 1,2,3 and 7, Risks: 4,5 and 6

c. Benefits: 2,3,6 and 7. Risks: 1,4 and 5

d. Benefits: 2,3,5 and 6. Risks: 1,4 and 7

Question 10 Which of the following encourages objective testing?

a. Unit testing

b. System testing

c. Independent testing

d. Destructive testing

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

IIIEvaluating exit criteria and reporting

IVPlanning 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 HI and II.

Question 13 Test objectives vary between projects and so must 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 test activities are supported by test data preparation tools?

a. Test management and control

b. Test specification and design

c. Test execution and logging

d. Performance and monitoring

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 program. If you don't 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

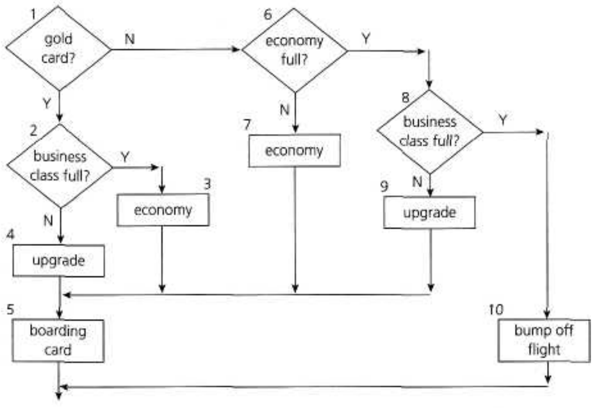


FIGURE 7.1 Control flow diagram for flight check-in

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 Modeling 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,XandY

b. VYandZ

c. V,WandZ

d. X,YandZ

Question 17 What is a test condition?

a. An input, expected outcome, precondition and postcondition

b. The steps to be taken to get the system to a given point

c. Something that can be tested

d. A specific state of the software, e.g. before a test can be run

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 Which of these statements about functional testing is true?

a. Structural testing is more important than functional testing as it addresses the code.

b. Functional testing is useful throughout the life cycle and can be applied by business analysts, testers, developers and users.

c. Functional testing is more powerful than static testing as you actually run the system and see what happens.

d. Inspection is a form of functional testing.

Question 21 What is the purpose of confirmation testing?

a. To confirm the users' confidence that the system will meet their business needs.

b. To confirm that a defect has been fixed correctly.

c. To confirm that no unexpected changes have been introduced or uncovered as a result of changes made.

d. To confirm that the detailed logic of a component conforms to its specification.

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 Which of the following best describes integration testing?

a. Testing performed to expose faults in the interfaces and in the interaction between integrated components.

b. Testing to verify that a component is ready for integration.

c. Testing to verify that the test environment can be integrated with the product.

d. Integration of automated software test suites with the product.

Question 24 According to the ISTQB Glossary, debugging:

a. Is part of the fundamental testing process.

b. Includes the repair of the cause of a failure.

c. Involves intentionally adding known defects.

d. Follows the steps of a test procedure.

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.

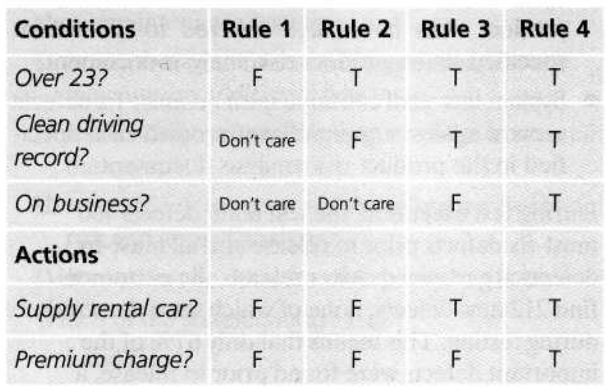
Question 26 Assume postal rates for 'light letters' are:

$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

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

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.

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

TCI: 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. TCI: Don't supply car; TC2: Supply car with premium charge.

b. TCI: Supply car with premium charge; TC2: Supply car with no premium charge.

c. TCI: Don't supply car; TC2: Supply car with no premium charge.

d. TCI: Supply car with premium charge; TC2: Don't 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 What does it mean if a set of tests has achieved 90% statement coverage?

a. 9 out of 10 decision outcomes have been exercised by this set of tests.

b. 9 out of 10 statements have been exercised by this set of tests.

c. 9 out of 10 tests have been run on this set of software.

d. 9 out of 10 requirements statements about the software are correct.

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 Requirement 24.3. A 'Postage Assistant' will calculate the amount of postage due for letters and small packages up to 1 kilogram in weight. The inputs are: the type of item (letter, book or other package) and the weight in grams. Which of the following conform to the required contents of a test case?

a. Test the three types of item to post and three different weights [Req 24.3]

b. Test 1: letter, 10 grams, postage €0.25. Test 2: book, 500 grams, postage €1.00. Test 3: package, 999 gram, postage €2.53 [Req 24.3]

c. Test 1: letter, 10 grams to Belgium. Test 2: book 500 grams to USA. Test 3: package, 999 grams to South Africa [Req 24.3]

d. Test 1: letter 10 grams, Belgium, postage €0.25. Test 2: package 999 grams to South Africa, postage €2.53

Question 32 What is the best description of static analysis?

a. The analysis of batch programs

b. The reviewing of test plans

c. The analysis of program code or other software artifacts

d. The use of black-box testing

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 according to which test principle?

a. Impossibility of exhaustive testing.

b. Importance of early testing.

c. The absence of errors fallacy.

d. Defect clustering.

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.

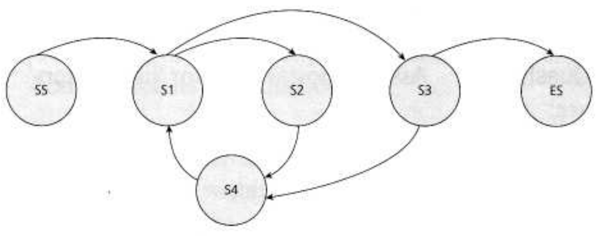

$$\begin{tikzcd}
SS \arrow[r, bend left] & S1 \arrow[r, bend left] \arrow[rr, bend left] & S2 \arrow[d] & S3 \arrow[r, bend left] \arrow[dl, bend left] & ES \\
 & & S4 \arrow[ul] & &
\end{tikzcd}$$

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 A test plan included the following clauses among the exit criteria:

- System test shall continue until all significant product risks have been covered to the extent specified in the product risk analysis document.
- System test shall continue until no must-fix defects remain against any significant product risks speci fied in the product risk analysis document.

During test execution, the test team detects 430 must-fix defects prior to release and all must-fix defects are resolved. After release, the customers find 212 new defects, none of which were detected during testing. This means that only 67% of the important defects were found prior to release, a percentage which is well below average in your industry. You are asked to find the root cause for the high number of field failures. Consider the following list of explanations:

I Not all the tests planned for the significant product risks were executed.

II The organization has unrealistic expectations of the percentage of defects that testing can find.

III A version-control issue has resulted in the release of a version of the software that was used during early testing.

IVThe product risk analysis failed to identify all the important risks from a customer point of view.

V The product risk analysis was not updated during the project as new information became available.

Which of the following statements indicate which explanations are possible root causes?

a. II, **III** and IV are possible explanations, but I and
V are not possible.

b. All five are possible explanations.

c. I, IV and V are possible explanations, but II and
III are not possible.

d. Ill, IV and V are possible explanations, but I and
II are not possible.

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 Consider the following statements about maintenance testing:

I It requires both re-test and regression test and
may require additional new tests.

II It is testing to show how easy it will be to maintain
the system.

IIIIt is difficult to scope and therefore needs careful
risk and impact analysis.

IVIt need not be done for emergency

bug fixes. Which of the statements are true?

a. **I** and **III**

b. **I** and IV

c. **II** and **III**

d. **II** and **IV**

Question 39 Which two specification-based testing techniques are most closely related to each other?

a. Decision tables and state transition testing

b. Equivalence partitioning and state transition
testing

c. Decision tables and boundary value analysis

d. Equivalence partitioning and boundary value analysis

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

a. Independent testers don't have to spend time
communicating with the project team.

b. Programmers 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.

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 paper in Chapter 7.

If you get any of the questions wrong or if you weren't 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 4 in Chapter 1 wrong, then go to Section 1.2 and read the first learning objective. Then re-read the part of the chapter that deals with that topic.

CHAPTER 1 FUNDAMENTALS OF TESTING

Question	Answer	Learning objective
1	A	1.1.3
2	B	1.1.5
3	C	1.1.5
4	A	1.2.1
5	A	1.1.6 and 1.2.3
6	C	1.4.1
7	B	1.5.1
8	D	1.1.6, 1.2.3, 1.3.1 and 1.4.1

CHAPTER 2 TESTING THROUGHOUT THE SOFTWARE LIFE CYCLE

Question	Answer	Learning objective
1	D	2.1.3
2	D	2.1.3
3	B	2.3.1
4	B	2.3.3
5	C	2.3.1
6	D	2.4.3
7	C	2.3.5
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	3.3.1
3	D	3.2.2
4	A	3.2.2
5	D	3.2.2
6	B	3.2.2
7	A	3.3.1
8	C	3.1.2
9	C	3.3.2

CHAPTER 4 TEST DESIGN TECHNIQUES

Question	Answer	Learning objective
1	D	4.1.1
2	A	4.1.4
3	C	4.1.5
4	A	4.2.1
5	B	4.2.2
6	C	4.3.1
7	D	4.3.1
8	B	4.3.1
9	B	4.3.2
10	C	4.4.2
11	A	4.3.2
12	C	4.4.2
13	C	4.4.4
14	A	4.5.1
15	D	4.5.2
16	B	4.6.1
17	A	4.3.1

CHAPTER 5 TEST MANAGEMENT

Question	Answer	Learning objective
1	B	5.1.1
2	D	5.1.4
3	B	5.1
4	A	5.2.2
5	C	5.2.3
6	C	5.2.5
7	D	5.2.7
8	A	5.2
9	C	5.3.1
10	B	5.3.2
11	A	5.3.3
12	C	5.4.1
13	D	5.5.1
14	B	5.5.2
15	A	5.5.3
16	A	5.5.5
17	B	5.5
18	D	5.6.1
19	C	5.6
20	A	Cross-section: 5.3, 5.5, 5.6

CHAPTER 6 TOOL SUPPORT FOR TESTING

Question	Answer	Learning objective
1	D	6.1.1
2	C	6.1.1
3	B	6.2.1
4	A	6.2.1
5	A	6.2.2
6	B	6.3.1
7	D	6.3.2

CHAPTER 7 MOCK EXAM

Question	Answer	Learning objective
1	B	4.2.2
2	A	1.3
3	A	2.1.3
4	D	4.1.5 and 5.5.4
5	C	5.5.4
6	B	2.3.5
7	C	4.4.2
8	B	5.6.2
9	C	6.2.1
10	C	1.5.1
11	D	3.1.2
12	B	1.4.1
13	A	1.5.2
14	B	6.1.1
15	A	4.4.3
16	A	6.1.2
17	C	4.1.2
18	B	5.2.4
19	A	4.3.1
20	B	2.3.2
21	B	2.3.5
22	B	6.3.3
23	A	2.2
24	B	1.1.5
25	C	1.1.2
26	C	4.3.1
27	C	4.3.1
28	D	4.5.1
29	B	4.4.1
30	C	5.2.1
31	B	4.1.3
32	C	3.3.1
33	D	1.2 and 1.4
34	D	5.4.1
35	C	4.3.1
36	C	5.2 and 5.5
37	B	3.2.3
38	A	2.4.3
39	D	4.3.1
40	D	5.1.2

THOMSON

Foundations of Software Testing

ISTQB Certification

Dorothy Graham, Erik Van Veenendaal, Isabel Evans, Rex Black

Visit the website at: **www.thomsonlearning.co.uk/istqb**