Test completion work products give closure to the whole of the test process and should provide ongoing ideas for increasing the effectiveness and efficiency of testing within the organization in the future.

1.4.4 Traceability between the test basis and test work products

We have mentioned bi-directional **traceability** for all test work products covered in Section 1.4.3. Bi-directional means that we can trace, for example, a given requirement through test conditions and test cases to the test execution results, but we can also trace the test execution results back to test cases, test cases back to test conditions, and test conditions back to requirements. No matter what the work products are called, this cross-linking gives many benefits to the test process. We saw how traceability can aid in the measurement and reporting of test coverage in order to report coverage and defects related to requirements, which is more meaningful and of more value to stakeholders.

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

Good traceability also supports the following:

- analyzing the impact of changes, whether to requirements or to the component or system
- making testing auditable, and being able to measure coverage
- meeting IT governance criteria (where applicable)
- improving the coherence of test progress reports and test summary reports to stakeholders, as described above
- relating the technical aspects of testing to stakeholders in terms that they can understand
- providing information to assess product quality, process capability and project progress against business goals.

You may find that your test management or requirements management tool provides support for traceability of work products; if so, make use of that feature. Some organizations find that they have to build their own management systems in order to organize test work products in the way that they want and to ensure that they have bi-directional traceability. However the support is implemented, it is important to have automated support for traceability – it is not something that can be sustained without tool support.

1.5 THE PSYCHOLOGY OF TESTING

SYLLABUS LEARNING OBJECTIVES FOR 1.5 THE PSYCHOLOGY OF TESTING (K2)

FL-1.5.1 **Identify the psychological factors that influence the success of testing (K1)**

FL-1.5.2 **Explain the difference between the mindset required for test activities and the mindset required for development activities (K2)**

1.5.1 Human psychology and testing

In this section, we'll discuss the various psychological factors that influence testing and its success. We'll also contrast the mindset of a tester and a developer. For a good introduction to the psychology of testing see Weinberg [2008].

As mentioned earlier, the ISTQB definition of software testing includes both dynamic testing and static testing. Let's review the distinction again. Dynamic software testing involves actually executing the software or some part of it, such as checking an application-produced report for accuracy or checking response time to user input. Static software testing does not execute the software but uses two possible approaches: automated static analysis on the code (e.g. evaluating its complexity) or on a document (e.g. to evaluate the readability of a use case); or reviews of code or documents (e.g. to evaluate a requirement specification for consistency, ambiguity and completeness).

Finding defects

While dynamic and static testing are very different types of activities, they have in common their ability to find defects. Static testing finds defects directly, while dynamic testing finds evidence of a defect through a failure of the software to behave as expected. Either way, people carrying out static or dynamic tests must be focused on the possibility – indeed, the high likelihood in many cases – of finding defects. Indeed, finding defects is often a primary objective of static and dynamic testing activities.

Identifying defects may unfortunately be perceived by developers as a criticism, not only of the product but also of its author – and in a sense, it is. But finding defects in testing should be *constructive* criticism, where testers have the best interest of the developer in mind. One meaning of the word 'criticism' is 'an examination, interpretation, analysis or judgement about something' – this is an objective assessment. But other meanings include disapproval by pointing out faults or shortcomings and even an attack on someone or something. Testing does not want to be the latter sense of criticism, but even when intended in the first sense, it can be perceived in the other ways. Testers need to be diplomats, along with everything else.

Bias

However, there is another factor at work when we are reporting defects; the author (developer) believes that their code is correct – they obviously did not write it to be intentionally wrong. This confidence in their understanding is in some sense necessary for developers; they cannot proceed without it. But at the same time, this confidence creates confirmation bias. Confirmation bias makes it difficult to accept information that disagrees with your currently held beliefs. Simply put, the author of a work product has confidence that they have solved the requirements, design, metadata or code problem, at least in an acceptable fashion; however, strictly speaking, that is false confidence. Other biases may also be at work, and it is also human nature to blame the bearer of bad news (which defects are perceived to be).

Testers are not always aware of their biases either, and they do have biases of their own. Since those biases are different from the developers, that is a benefit, but a lack of awareness of those biases sets up potential conflict.

This reluctance to accept that their work is not perfect is why some people regard testing as a destructive activity (trying to destroy their work) rather than the constructive activity it is (trying to construct better quality software). Good testing contributes greatly to product quality and project quality, as we saw in Sections 1.1 and 1.2.

While some developers are aware of their biases when they participate in reviews and perform unit testing of their own work products, those biases act to impede their effectiveness at finding their own defects. The mental mistakes that caused them to create the defects remain in their minds in most cases. When proofreading our own work, for example, we see what we meant, not what we wrote.

Review and test your own work?

Should software work product developers – business analysts, system designers, architects, database administrators and developers – review and test their own work? They certainly should; they have a deep understanding about the system, and quality is everyone's responsibility.

However, many business analysts, system designers, architects, database administrators and developers do not know the review, static analysis and dynamic testing techniques discussed in the Foundation Syllabus and this book. While that situation is gradually changing, much of the self-testing by software work product developers is either not done or is not done as effectively as it could be. The principles and techniques in the Foundation Syllabus and this book are intended to help either testers or others to be more effective at finding defects, both their own and those of others.

Attitudes

It is a particular problem when a tester revels in being the bearer of bad news. For example, one tester made a revealing – and not very flattering – remark during an interview with one of the authors. When asked what he liked about testing, he responded, 'I like to catch the developers'. He went on to explain that, when he found a defect in someone's work, he would go and demonstrate the failure on the programmer's workstation. He said that he made sure that he found at least one defect in everyone's work on a project, and went through this process of ritually humiliating the programmer with each and every one of his colleagues. When asked why, he said, 'I want to prove to everyone that I am their intellectual equal'. This person, while possessing many of the skills and traits one would want in a tester, had exactly the wrong personality to be a truly professional tester.

Instead of seeing themselves as their colleagues' adversaries or social inferiors out to prove their equality, testers should see themselves as teammates. In their special role, testers provide essential services in the development organization. They should ask themselves, 'Who are the stakeholders in the work that I do as a tester?' Having identified these stakeholders, they should ask each stakeholder group, 'What services do you want from testing, and how well are we providing them?'

While the specific services are not always defined, it is common that mature and wise developers know that studying their mistakes and the defects they have introduced is the key to learning how to get better. Further, smart software development managers understand that finding and fixing defects during testing not only reduces the level of risk to the quality of the product, it also saves time and money when compared to finding defects in production.

Communication

Clearly defined objectives and goals for testing, combined with constructive styles of communication on the part of test professionals, will help to avoid most negative personal or group dynamics between testers and their colleagues in the development team. Whenever defects are found, true testing professionals distinguish themselves by demonstrating good interpersonal skills. True testing professionals communicate

facts about defects, progress and risks in an objective and constructive way that counteracts these misperceptions as much as possible. This helps to reduce tensions and build positive relationships with colleagues, supporting the view of testing as a constructive and helpful activity. While this is not necessary, we have noticed that many consummate testing professionals have business analysts, system designers, architects, developers and other specialists with whom they work as close personal friends.

This applies not only to testers but also to test managers, and not just to defects and failures but to all communication about testing, such as test results, test progress and risks.

Having good communication skills is a complex topic, well beyond the scope of a book on fundamental testing techniques. However, we can give you some basics for good communication with your development colleagues:

- Remember to think of your colleagues as teammates, not as opponents or adversaries. The way you regard people has a profound effect on the way you treat them. You do not have to think in terms of kinship or achieving world peace, but you should keep in mind that everyone on the development team has the common goal of delivering a quality system, and everyone must work together to accomplish that. Start with collaboration, not battles.
- Make sure that you focus on and emphasize the value and benefits of testing. Remind your developer colleagues that defect information provided by testing can help them to improve their own skills and future work products. Remind managers that defects found early by testing and fixed as soon as possible will save time and money and reduce overall product quality risk. Also, be sure to respond well when developers find problems in your own test work products. Ask them to review them and thank them for their findings (just as you would like to be thanked for finding problems in their work).
- Recognize that your colleagues have pride in their work, just as you do, and as such you owe them a tactful communication about defects you have found. It is not really any harder to communicate your findings, especially the potentially embarrassing findings, in a neutral, fact-focused way. In fact, you will find that if you avoid criticizing people and their work products, but instead keep your written and verbal communications objective and factual, you will also avoid a lot of unnecessary conflict and drama with your colleagues.
- Before you communicate these potentially embarrassing findings, mentally put yourself in the position of the person who created the work product. How are they going to feel about this information? How might they react? What can you do to help them get the essential message that they need to receive without provoking a negative emotional reaction from them?
- Keep in mind the psychological element of cognitive dissonance. Cognitive dissonance is a defect – or perhaps a feature – in the human brain that makes it difficult to process unexpected information, especially bad news. So, while you might have been clear in what you said or wrote, the person on the receiving end might not have clearly understood. Cognitive dissonance is a two-way street, too, and it is quite possible that you are misunderstanding someone's reaction to your findings. So, before assuming the worst about someone and their motivations, confirm that the other person has understood what you have said and vice versa.

1.5.2 Tester's and developer's mindsets

Testers and developers actually have different thought processes and different objectives for their work. A mindset reflects an individual's assumptions and the way that they like to solve problems and make decisions.

A tester's mindset should include the following:

- Curiosity. Good testers are curious about why systems behave the way they do and how systems are built. When they see unexpected behaviour, they have a natural urge to explore further, to isolate the failure, to look for more generalized problems and to gain deeper understanding.
- Professional pessimism. Good testers expect to find defects and failures. They understand human fallibility and its implications for software development. (However, this is not to say that they are negative or adversarial, as we'll discuss in a moment.)
- A critical eye. Good testers couple this professional pessimism with a natural inclination to doubt the correctness of software work products and their behaviours as they look at them. A good tester has, as a personal slogan, 'If in doubt, it is a bug'.
- Attention to detail. Good testers notice everything, even the smallest details. Sometimes these details are cosmetic problems like font-size mismatches, but sometimes these details are subtle clues that a serious failure is about to happen. This trait is both a blessing and a curse. Some testers find that they cannot turn this trait off, so they are constantly finding defects in the real world – even when not being paid to find them.
- Experience. Good testers not only know a defect when they see one, they also know where to look for defects. Experienced testers have seen a veritable parade of bugs in their time, and they leverage this experience during all types of testing, especially experience-based testing such as error guessing (see Chapter 4).
- Good communication skills. All of these traits are essential, but, without the ability to effectively communicate their findings, testers will produce useful information that will, alas, be put to no use. Good communicators know how to explain the test results, even negative results such as serious defects and quality risks, without coming across as preachy, scolding or defeatist.

A good tester has the skills, the training, the certification and the mindset of a professional tester, and of these four skills, the most important – and perhaps the most elusive – is the mindset.

The tester's mindset is to think about what could go wrong and what is missing. The tester looks at a statement in a requirement or user story and asks, 'What if it isn't? What haven't they thought of here? What could go wrong?' That mindset is quite different from the mindset that a business analyst, system designer, architect, database administrator or developer must bring to creating the work products involved in developing software. While the testers (or reviewers) must assume that the work product under review or test is defective in some way – and it is their job to find those defects – the people developing that work product must have confidence that they understand how to do so properly. Looking at a statement in a requirement or user story, the developer thinks, 'How can I implement this? What technical challenges do I need to solve?'

Having a tester or group of testers who are organizationally separate from development, either as individuals or as an independent test team, can provide significant benefits, such as increased defect-detection percentage. A tester's mindset is a 'different pair of eyes' and independent testers can see things that developers do not see (because of confirmation bias discussed above). This is especially important for large, complex or safety-critical systems.

However, independence from the developers does not mean an adversarial relationship with them. In fact, such a relationship is toxic, often fatally so, to a test team's effectiveness.

The softer side of software testing is often the harder side to master. A tester may have adequate or even excellent technique skills and certifications, but if they do not have adequate interpersonal and communication skills, they will not be an effective tester. Such soft skills can be improved with training and practice. The best testers continuously strive to attain a more professional mindset, and it is a lifelong journey.

CHAPTER REVIEW

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

From Section 1.1, you should now know what testing is. You should be able to remember the typical objectives of testing. You should know the difference between testing and debugging. You should know the Glossary keyword terms **debugging**, **test object**, **test objective**, **testing**, **validation** and **verification**.

From Section 1.2, you should now be able to explain why testing is necessary and support that explanation with examples. You should be able to explain the difference between testing and quality assurance and how they work together to improve quality. You should be able to distinguish between an error (made by a person), a defect (in a work product) and a failure (where the component or system does not perform as expected). You should know the difference between the root cause of a defect and the effects of a defect or failure. You should know the Glossary terms **defect**, **error**, **failure**, **quality**, **quality assurance** and **root cause**.

You should be able to explain the seven principles of testing, discussed in Section 1.3.

From Section 1.4, you should now recognize a test process. You should be able to recall the main testing activities of test planning, test monitoring and control, test analysis, test design, test implementation, test execution and test completion. You should be familiar with the work products produced by each test activity. You should know the Glossary terms **coverage**, **test analysis**, **test basis**, **test case**, **test completion**, **test condition**, **test control**, **test data**, **test design**, **test execution**, **test execution schedule**, **test implementation**, **test monitoring**, **test oracle**, **test planning**, **test procedure**, **test suite**, **testware** and **traceability**.

From Section 1.5, you now should be able to explain the psychological factors that influence the success of testing. You should be able to explain and contrast the mindsets of testers and developers, and why these differences can lead to problems.