1.5 The Psychology of Testing

In this section, we'll discuss the various psychological factors that influence testing and its success. These include clear objectives for testing, the proper roles and balance of self-testing and independent testing, clear, courteous communication and feedback on defects. We'll also contrast the mindset of a tester and of a developer. You'll find a single Syllabus term in this section, independent testing, and the glossary term, independence.

1.5.1 Independent testing - who is a tester? The mindset we want to use while testing and reviewing is different from the one we use while analyzing or developing. By this we mean that, if we are building something we are working positively to solve problems in the design and to realize a product that meets some need. However, when we test or review a product, we are looking for defects in the product and thus are critical of it. Suppose you were going to cook a meal to enter in a competition for chefs. You select the menu, collect the ingredients, cook the food, set the table, and serve the meal. If you want to win, you do each task as well as you can. Suppose instead you are one of the judges evaluating the competition meals. You examine everything critically, including the menu, the ingredients, the methods used, keeping to time and budget allowances, choice of ingredients, the elegance of the table setting and the serving, and the look and taste of the meal. To differentiate between the competition chefs, you'll praise every good aspect of their performances but you'll also note every fault and error each chef made. So it is with software testing: building the software requires a different mindset from testing the software. We do not mean that a tester cannot be a programmer, or that a programmer cannot be a tester, although they often are separate roles. In fact, programmers are testers - they test the components which they build, and the integration of the components into the system. The good chef will be as critical as the competition judges of his own work, in order to prevent and rectify errors and defects before anyone notices them. So, with the right mindset, programmers can test their own code; indeed programmers do test their own code and find many problems, resolving them before anyone else sees the code. Business analysis and marketing staff should review their own requirements. System architects should review their own designs. However, we all know it is difficult to find our own mistakes. So, business analysts, marketing staff, architects and programmers often rely on others to help test their work. This other person might be a fellow analyst, designer or developer. A person who will use the software may help test it. Business analysts who worked on the requirements and design may perform some tests. Testing specialists - professional testers - are often involved. In fact, testing may involve a succession of people each carrying out a different level of testing. This allows an independent test of the system. We'll look at the points in the software development life cycle where testing takes place in

Chapter 2. You'll see there that several stages of reviews and testing are carried out throughout the life cycle and these may be independent reviews and tests. Early in the life cycle, reviews of requirements and design documents by someone other than the author helps find defects before coding starts and helps us build the right software. Following coding, the software can be tested independently.

This degree of independence avoids author bias and is often more effective at finding defects and failures. Several levels of independence can be identified, listed here from the lowest level of independence to the highest:

• tests by the person who wrote the item under test;

• tests by another person within the same team, such as another programmer;

• tests by a person from a different organizational group, such as an independ ent test team;

• tests designed by a person from a different-organization or company, such as outsourced testing or certification by an external body. We should note, however, that independence is not necessarily the most important factor in good testing. Developers who know how to test and who are, like good chefs, self-critical, have the benefit of familiarity and the prideof-work that comes with true professionalism. Such developers can efficiently find many defects in their own code. Some software development methodologies insist on developers designing tests before they start coding and executing those tests continuously as they change the code. This approach promotes early testing and early defect detection, which is cost effective. Remember, independent testing may be carried out at any level of testing and the choice of independence level depends on the risk in the particular context.

1.5.2 Why do we sometimes not get on with the rest of the team? As well as independence, separation of the tester role from the developer role is also done to help focus effort and to provide the benefits of trained and professional testing resources. In many organizations, earlier stages of testing are carried out by the developers and integrators and later stages independently, either by a specialist test group or by the customers. However, this separation can lead to problems as well as advantages. The advantage of independence and focus may be lost if the inter-team relationships deteriorate, as we'll see. Each organization and each project will have its own goals and objectives. Different stakeholders, such as the customers, the development team and the managers of the organization, will have different viewpoints about quality and have their own objectives. Because people and projects are driven by objectives, the stakeholder with the strongest views or the greatest influence over a group will define, consciously or subconsciously, what those

objectives are. People tend to align their plans with these objectives. For example, depending on the objective, a tester might focus either on finding defects or on confirming that software works. But if one stakeholder is less influential during the project but more influential at delivery, there may be a clash of views about whether the testing has met its objectives. One manager may want the confirmation that the software works and that it is 'good enough' if this is seen as a way of delivering as fast as possible. Another manager may want the testing to find as many defects as possible before the software is released, which will take longer to do and will require time for fixing, re-testing and regression testing. If there are not clearly stated objectives and exit criteria for testing which all the stakeholders have agreed, arguments might arise, during the testing or after release, about whether 'enough' testing has been done. Many of us find it challenging to actually enjoy criticism of our work. We usually believe that we have done our best to produce work (documents, code, tests, whatever) which is correct and complete. So when someone else identifies a defect, a mistake we have made, we might take this personally and get annoyed with the other person, especially if we are under time pressure. This is true of managers, staff, testers and developers. We all make mistakes and we sometimes get annoyed, upset or depressed when someone points them out. So,

when as testers we run a test which (from our viewpoint) is a good test that finds defects and failures in the software, we need to be careful how we react. We are pleased, of course, since we have found a good bug! But how will the requirements analyst, designer, developer, project manager and customer react? The people who build products may react defensively and perceive this reported defect as personal criticism against the product and against the author. The project manager may be annoyed with everyone for holding up the project. The customer may lose confidence in the product because he can see defects. Because testing can be seen as a destructive activity, we need to take care to report on defects and failures as objectively and politely as possible.

If others are to see our work as constructive in the management of product risks, we need to be careful when we are reviewing and when we are testing:

• Communicate findings on the product in a neutral, fact-focused way without criticizing the person who created it. For example, write objective and factual incident reports and review findings. - Don't gloat - you are not perfect either! - Don't blame - any mistakes are probably by the group rather than an individual. - Be constructively critical and discuss the defect and how you are going to log it.

• Explain that by knowing about this now we can work round it or fix it so the delivered system is better for the customer. - Say what you liked and what worked, as well as what didn't work. - Show what the risk is honestly - not everything is high

priority. - Don't just see the pessimistic side - give praise as well as criticism. - Show what risks have been uncovered and the benefits of the review or test.

• Start with collaboration rather than battles. Remind everyone of the common goal of better quality systems. - Be polite and helpful, collaborate with your colleagues. - Try to understand how the other person feels and why they react as they do. - Confirm that the other person has understood what you have said and vice versa. - Explain how the test or review helps the author - what's in it for him or her. - Offer your work to be reviewed, too. It's our job as reviewers and testers to provide everyone with clear, objective information and to do this we go bug-hunting, defect-mining and failuremaking. People who will make good reviewers and testers have the desire and ability to find problems, and this is true whether testing is their main job or part of their role as a developer. These people build up experience of where errors are likely to be made, and are characterized by their curiosity, professional pessimism, critical eye and attention to detail. However, unless we also have good interpersonal and communication skills, courtesy, understanding of others and a good attitude towards our peers, colleagues, customers, managers and the rest of the team, we will fail as testers because no-one will listen to us.

The tester and test leader need good interpersonal skills to communicate factual information about defects, progress and risks in a constructive way [Perry]. For the author of the software or document, defect information can help them improve their skills, but only if it is provided in a way that helps them. One book that you might find interesting in this context is Six Thinking Hats [de Bono]. It is not about testing but describes a way to communicate different information: facts; our emotions; pessimistic and optimistic thoughts; and creative ideas. When reviewing or testing, we need to communicate facts objectively, but the other types of information are useful too: 'This happened; this is how I felt about it; this is what was good; this is what might go wrong; here is a way we could solve the problem'. As part of supplying the risk assessment, we can help the managers and customers make risk-based decisions based on the cost and time impact of a defect. If we test and find a defect that would cost $15 000 to fix and re-test/regression test, is it worth fixing? If it would cause a business impact of $50 000 in the live environment the customer may want it fixed. If it has a potential business impact of $10 000 but any fix is difficult to do and likely to have adverse impact elsewhere, it may be better not to fix. By providing the team with information about the defect in terms they find useful, we can help them to make the right decision about fixing or not fixing the problems. Generally we say that defects found and fixed during testing will save time and money later and reduce risks, so we need to show that is the case in order for the testing to be valued. To help you think about the psychology of testing, there is an exercise at the end of the chapter, following the practice examination question.