CHAPTER 3

The Testing Team

The capabilities of the testing team can greatly affect the success, or failure, of the testing effort. An effective testing team includes a mixture of technical and domain expertise relevant to the software problem at hand. It is not enough for a testing team to be technically proficient with the testing techniques and tools necessary to perform the actual tests. Depending on the complexity of the domain, a test team should also include members who have a detailed understanding of the problem domain. This knowledge enables them to create effective test artifacts and data and to effectively implement test scripts and other test mechanisms.

In addition, the testing team must be properly structured, with defined roles and responsibilities that allow the testers to perform their functions with minimal overlap and without uncertainty regarding which team member should perform which duties. One way to divide testing resources is by specialization in particular application areas and nonfunctional areas. The testing team may also have more role requirements than it has members, which must be considered by the test manager.

As with any team, continual evaluation of the effectiveness of each test team member is important to ensuring a successful test effort. Evaluation of testers is performed by examining a variety of areas, including the types of defects generated, and the number and types of defects missed. It is never good practice to evaluate a test engineer's performance using numbers of defects generated alone, since this metric by itself does not tell the whole story. Many factors must be considered during this type of evaluation, such as complexity of functionality tested, time constraints, test

engineer role and responsibilities, experience, and so on. Regularly evaluating team members using valid criteria makes it possible to implement improvements that increase the effectiveness of the overall effort.

Item 13: Define Roles and Responsibilities[1]

Test efforts are complex, and require that the test team possess a diversity of expertise to comprehend the scope and depth of the required test effort and develop a strategy for the test program.

In order for everyone on the test team to be aware of what needs to get done and who will take the lead on each task, it is necessary to define and document the roles and responsibilities of the test-team members. These should be communicated, both verbally and in writing, to everyone on the team. Identifying the assigned roles of all test-team members on the project enables everyone to clearly understand which individual is responsible for each area of the project. In particular, it allows new team members to quickly determine whom to contact if an issue arises.

In order to identify the individuals needed to perform a particular task, a task description should be created. Once the scope of the task is understood, it will be easier to assign particular team members to the task.

To help ensure successful execution of the task, **work packages** can be developed and distributed to the members of the test team. Work packages typically include the organization of the tasks, technical approach, task schedule, spending plan, allocation of hours for each individual, and a list of applicable standards and processes.

The number of test-engineering roles in a project may be greater than the number of test-team members. (The roles required depend on the task at hand, as discussed

1. Adapted from Elfriede Dustin et al., *Automated Software Testing* (Reading, Mass.: Addison-Wesley, 1999), Table 5.11, 183–186.

in Chapter 2.) As a result, a test engineer may "wear many hats," being responsible for more than one role.

Table 13.1 shows some example responsibilities and skills required for each test-program role.

Table 13.1—Test Program Roles

Test Manager

Responsibilities	Skills
▪ Liaison for interdepartmental interactions: Representative of the testing team ▪ Customer interaction, if applicable ▪ Recruiting, staff supervision, and staff training ▪ Test budgeting and scheduling, including test-effort estimations	▪ Understands testing process or methodology ▪ Familiar with test-program concerns including test environment and data management, trouble reporting and resolution, and test design and development ▪ Understands manual testing techniques and automated testing best practices
▪ Test planning, including development of testing goals and strategy ▪ Vendor interaction ▪ Test-tool selection and introduction	▪ Understands application business area, application requirements
▪ Cohesive integration of test and development activities	▪ Skilled at developing test goals, objectives, and strategy
▪ Acquisition of hardware and software for test environment	▪ Familiar with different test tools, defect-tracking tools, and other test-support COTS tools and their use
▪ Test environment and test product configuration management.	▪ Good at all planning aspects, including people, facilities, and schedule
▪ Test-process definition, training and continual improvement	
▪ Use of metrics to support continual test-process improvement	
▪ Test-program oversight and progress tracking	
▪ Coordinating pre- and post-test meetings	

Test Lead

Responsibilities	Skills
▪ Technical leadership for the test program, including test approach	▪ Understands application business area and application requirements
▪ Support for customer interface, recruiting, test-tool introduction, test planning, staff supervision, and cost and progress status reporting	▪ Familiar with test-program concerns including test-data management, trouble reporting and resolution, test design, and test development
▪ Verifying the quality of the requirements, including testability, requirement definition, test design, test-script and test-data development, test automation, test-environment configuration; test-script configuration management, and test execution	▪ Expertise in a variety of technical skills including programming languages, database technologies, and computer operating systems
▪ Interaction with test-tool vendor to identify best ways to leverage test tool on the project	▪ Familiar with different test tools, defect-tracking tools, and other COTS tools supporting the testing life cycle, and their use
▪ Staying current on latest test approaches and tools, and transferring this knowledge to test team	
▪ Conducting test-design and test-procedure walk-throughs and inspections	
▪ Implementing test-process improvements resulting from lessons learned and benefits surveys	
▪ Test Traceability Matrix (tracing the test procedures to the test requirements)	
▪ Test-process implementation	
▪ Ensuring that test-product documentation is complete	

Usability[2] Test Engineer

Responsibilities	Skills
▪ Designs and develops usability testing scenarios ▪ Administers usability testing process	▪ Proficient in designing test suites ▪ Understanding usability issues
▪ Defines criteria for performing usability testing, analyzes results of testing sessions, presents results to development team	▪ Skilled in test facilitation
▪ Develops test-product documentation and reports	▪ Excellent interpersonal skills
▪ Defines usability requirements, and interacts with customer to refine them	▪ Proficient in GUI design standards
▪ Participates in test-procedure walk-throughs	

Manual Test Engineer

Responsibilities	Skills
▪ Designs and develops test procedures and cases, with associated test data, based upon functional and nonfunctional requirements	▪ Has good understanding of GUI design
▪ Manually executes the test procedures	▪ Proficient in software testing
▪ Attends test-procedure walk-throughs	▪ Proficient in designing test suites
▪ Conducts tests and prepares reports on test progress and regression	▪ Proficient in the business area of application under test ▪ Proficient in testing techniques ▪ Understands various testing phases
▪ Follows test standards	▪ Proficient in GUI design standards

2. The term **usability** refers to the intuitiveness and ease-of-use of an application's user interface.

Automated Test Engineer (Automater/Developer)

Responsibilities	Skills
▪ Designs and develops test procedures and cases based upon requirements	▪ Good understanding of GUI design
▪ Designs, develops and executes reusable and maintainable automated scripts ▪ Uses capture/playback tools for GUI automation and/or develops test harnesses using a development or scripting language, as applicable	▪ Proficient in software testing
▪ Follows test-design standards	▪ Proficient in designing test suites
▪ Conducts/Attends test procedure walk-throughs	▪ Proficient in working with test tools
▪ Executes tests and prepares reports on test progress and regression	▪ Programming skills
▪ Attends test-tool user groups and related activities to remain abreast of test-tool capabilities	▪ Proficient in GUI design standards

Network Test Engineer

Responsibilities	Skills
▪ Performs network, database, and middleware testing	▪ Network, database and system administration skills
▪ Researches network, database, and middleware performance monitoring tools ▪ Develops load and stress test designs, cases, and procedures ▪ Supports walk-throughs or inspections of load and stress test procedures	▪ Expertise in a variety of technical skills, including programming languages, database technologies, and computer operating systems
▪ Implements performance monitoring tools on ongoing basis ▪ Conducts load and stress testing	▪ Product evaluation and integration skills ▪ Familiarity with network sniffers, and available tools for load and stress testing

Test Environment Specialist

Responsibilities	Skills
▪ Responsible for installing test tool and establishing test-tool environment	▪ Network, database and system administration skills
▪ Responsible for creating and controlling test environment by using environment setup scripts	▪ Expertise in a variety of technical skills, including programming and scripting languages, database technologies, and computer operating systems
▪ Creates and maintains test database (adds/restores/deletes, etc)	▪ Test tool and database experience
▪ Maintains requirements hierarchy within test-tool environment	▪ Product evaluation and integration skills

Security Test Engineer

Responsibilities	Skills
▪ Responsible for security testing of the application	▪ Understands security testing techniques ▪ Background in security ▪ Security test tool experience

Test Library and Configuration Specialist [a]

Responsibilities	Skills
▪ Test-script change management	▪ Network, database, and system administration skills
▪ Test-script version control	▪ Expertise in a variety of technical skills including programming languages, database technologies, and computer operating systems
▪ Maintaining test-script reuse library ▪ Creating the various test builds, in some cases	▪ Configuration-management tool expertise
	▪ Test-tool experience

a. This type of configuration is often done by a separate configuration management department.

It is important that the appropriate roles be assigned to the right people, based on their skill sets and other strengths. Individual testers can specialize in specific areas of the application, as well as nonfunctional areas. Assigning testers to specific areas of the application may enable them to become experts in those specific areas. In addition to focusing on particular functional testing areas, different team members should specialize in specific nonfunctional testing areas, performance testing, compatibility, security, and concurrency.

Teams that use automated test tools should include personnel with software-development skills. Automated testing requires that test scripts be developed, executed, and managed. For this reason, the skills and activities pertinent to performing manual testing differ from those required for performing automated software testing. Because of this difference, manual test roles should be listed separately in the definition of roles and responsibilities. This does not mean that an "automated test engineer" won't be expected to do some manual testing from time to time.

If multiple products are under test, testers can maximize continuity by keeping specific team members assigned to the same product domains for some period of time. It can be difficult for testers to switch between significantly different technical or problem domains too often. Testers should be kept on either the same type of products (e.g., Web vs. desktop applications) or in the same problem areas, depending on where their strengths lie. The most effective testers understand the business and system requirements, in addition to being technically adept.

Table 13.2 gives an example of a test-team structure. Basic skills are assumed and not listed. For example, if working in a Windows environment, it is assumed that the team members have Windows skills; if working in a UNIX environment, it is assumed the team members have basic UNIX skills.

Table 13.2 Example Test-Team Assignments

Position	Products	Duties / Skills	Roles and Responsibilities
Test Manager	Desktop Web	Responsible for test program, customer interface, test-tool introduction, and staff recruiting and supervision Skills: Management skills, MS Project, Winrunner, SQL, SQL Server, UNIX, VC++, Web applications, test-tool experience	Manage test program

(*continued*)

Table 13.2 Example Test-Team Assignments (cont.)

Position	Products	Duties / Skills	Roles and Responsibilities
Test Lead	Desktop Web	Staff supervision, cost/progress/test status reporting, and test planning, design, development, and execution Skills: TeamTest, Purify, Visual Basic, SQL, Winrunner, Robot, UNIX, MS Access, C/C++, SQL Server	[Reference the related testing requirements here] Develop automated test scripts for functional test procedures
Test Engineer	Desktop Web	Test planning, design, development, and execution Defect identification and tracking Skills: Test-tool experience, financial system experience	[Reference the related testing requirements here] Develop test harness
Test Engineer	Desktop Web	Test planning, design, development, and execution Defect identification and tracking Skills: Test-tool experience, financial system experience	Performance testing [Reference the related testing requirements here]
Test Engineer	Desktop	Test planning, design, development, and execution Defect identification and tracking Skills: Financial system experience	Configuration testing, installation testing [Reference the related testing requirements here]

(*continued*)

Table 13.2 Example Test-Team Assignments (cont.)

Position	Products	Duties / Skills	Roles and Responsibilities
Test Engineer	Web	Responsible for test tool environment, network, and middleware testing Performs all other test activities Defect identification and tracking Skills: Visual Basic, SQL, CNE, UNIX, C/C++, SQL Server	Security testing [Reference the related testing requirements here]
Jr. Test Engineer	Desktop	Performs test planning, design, development, and execution Defect identification and tracking Skills: Visual Basic, SQL, UNIX, C/C++, HTML, MS Access	[Reference the related testing requirements here]

Table 13.2 identifies test-team positions and their assignments on the project, together with the products they are working on. The duties that must be performed by the person in each of the positions are outlined, as are the skills of the personnel fulfilling those positions. Also noted are the products to which each position on the team is assigned.

Item 1 of this book emphasized the importance of the testing team's involvement from the beginning of the product's life cycle. If early involvement of testers has become an established practice in an organization, it is possible (and necessary) to define and document the role of the testing team during each phase of the life cycle, including the deliverables expected from the testing team upon completion of each phase.

Item 14: Require a Mixture of Testing Skills, Subject-Matter Expertise, and Experience

The most effective testing team consists of team members with a mixture of expertise, such as subject matter, technology, and testing techniques, plus a mixture of experience levels, such as beginners and expert testers. Subject-matter experts (SMEs) who understand the details of the application's functionality play an important role in the testing team.

The following list describes these concepts in more detail.

- *Subject matter expertise.* A technical tester might think it is feasible to learn the subject matter in depth, but this is usually not the case when the domain is complex. Some problem domains, such as tax law, labor contracts, and physics, may take years to fully understand. It could be argued that detailed and specific requirements should include all the complexities possible, so that the developer can properly design the system and the tester can properly plan the testing. Realistically, however, budget and time constraints often lead to requirements that are insufficiently detailed, leaving the content open to interpretation. Even detailed requirements often contain internal inconsistencies that must be identified and resolved.

  For these reasons, each SME must work closely with the developer and other SMEs on the program (for example, tax-law experts, labor-contracts experts, physicists) to parse out the intricacies of the requirements. Where there are two SMEs, they must be in agreement. If two SMEs cannot agree, a

third SME's input is required. A testing SME will put the final stamp of approval on the implementation, after appropriate testing.

- *Technical expertise.* While it is true that a thorough grasp of the domain is a valuable and desirable trait for a tester, the tester's effectiveness will be diminished without some level of understanding of software (including test) engineering. The most effective subject-matter expert testers are those who are also interested and experienced in the technology—that is, those who have taken one or more programming courses, or have some related technical experience. Subject-matter knowledge must be complemented with technical knowledge, including an understanding of the science of software testing.

  Technical testers, however, require a deeper knowledge of the technical platforms and architectural makeup of a system in order to test successfully. A technical tester should know how to write automated scripts; know how to write a test harness; and understand such technical issues as compatibility, performance, and installation, in order to be best prepared to test for compliance. While it is beneficial for SMEs to possess some of this knowledge, it is acceptable, of course, for them to possess a lower level of technical expertise than do technical testers.

- *Experience level.* A testing team is rarely made up exclusively of expert testers with years of expertise—nor would that necessarily be desirable. As with all efforts, there is room for apprentices who can be trained and mentored by more-senior personnel. To identify potential areas for training and advancement, the test manager must review the difference between the skill requirements and an individual's actual skills.

  A junior tester could be tasked with testing the lower-risk functionality, or cosmetic features such as the GUI interface controls (if this area is considered low-risk). If a junior tester is tasked with testing of higher-risk functionality, the junior tester should be paired with a more-senior tester who can serve as a mentor.

Although technical and subject-matter testers contribute to the testing effort in different ways, collaboration between the two types of testers should be encouraged. Just as it would take a technical tester a long time to get up to speed on all the details of the subject matter, it would take a domain or subject-matter expert a long time to become conversant with the technical issues to consider during testing. Cross-training should be provided to make the technical tester acquainted with the subject matter and the subject-matter expert familiar with the technical issues.

Some testing tasks may require specific skills within the technical or subject-matter area. For example, a tester who is experienced, or at least familiar, with usability testing techniques should be responsible for usability testing. A tester who is not skilled in this area can only guess what makes an application usable. Similarly, in the case of localization testing, an English speaker can only guess regarding the correct translation of a Web site into another language. A more effective localization tester would be a native speaker of the language into which the site has been translated.

Item 15: Evaluate the Tester's Effectiveness[1]

Maintaining an effective test program requires that the implementation of its elements, such as test strategy, test environment, and test-team make-up, be continuously evaluated, and improved as needed. Test managers are responsible for ensuring that the testing program is being implemented as planned and that specific tasks are being executed as expected. To accomplish this, they must track, monitor, and evaluate the implementation of the test program, so it can be modified as needed.

At the core of test-program execution are the test engineers. The ability of testers to properly design, document, and execute effective tests, accurately interpret the results, document any defects, and track them to closure is critical to the effectiveness of the testing effort. A test manager may plan the perfect testing process and select the ideal strategy, but if the test-team members do not effectively execute the testing process (for example, participating effectively in requirements inspections and design walk-throughs) and complete all strategic testing tasks as assigned (such as executing specific test procedures), important defects may be discovered too late in the development life cycle, resulting in increased costs. Worse, defects may be completely overlooked, and make their way into production software.

A tester's effectiveness can also make a big difference in relationships with other project groups. A tester who frequently finds bogus errors, or reports "user errors"

1. Adapted from Elfriede Dustin, "Evaluating a Tester's Effectiveness," *Stickyminds.com* (Mar. 11, 2002). See http://www.effectivesoftwaretesting.com.

when the application works as expected but the tester misunderstands the requirement, or (worst of all) often overlooks critical defects loses credibility with other team members and groups, and can tarnish the reputation of an entire test program.

Evaluating a tester's effectiveness is a difficult and often subjective task. Besides the typical elements in any employee's performance, such as attendance, attentiveness, attitude, and motivation, there are specific testing-related measures against which a tester can be evaluated. For example, all testers must be detail oriented and possess analytical skills, independent of whether they are technical testers, subject-matter experts, security testers, or usability testers.

The evaluation process starts with recruitment. The first step is to hire a tester with the skills required for the roles and responsibilities assigned to each position. (See Item 13 for a discussion on roles, responsibilities, and skills.)

In the case where a testing team is "inherited" rather than hired for the project, evaluation is more complicated. In such a case it is necessary for the manager to become familiar with the various testers' backgrounds, so the team members can be tasked and evaluated based on their experience, expertise, and backgrounds. It may become necessary to reassign some team members to other roles as their abilities become better known.

A test engineer's performance cannot be evaluated unless there are specified roles and responsibilities, tasks, schedules, and standards. The test manager must, first and foremost, state clearly what is expected of the test engineer, and by when.

Following is a typical list of expectations that must be communicated to testers.

- *Observe standards and procedures.* The test engineer must be aware of standards and procedures to be followed, and processes must be communicated. Standards and procedures are discussed in Item 21.
- *Keep schedules.* Testers must be aware of the test schedule, including when test plans, test designs, test procedures, scripts, and other testing products must be delivered. In addition, the delivery schedule of software components to testing should be known by all testers.
- *Meet goals and perform assigned tasks.* Tasks must be documented and communicated, and deadlines must be scheduled, for each tester. The test manager and the test engineer must agree on the assigned tasks.

- *Meet budgets.* For testers evaluating testing tools or other technology that must be purchased, the available budget must be communicated so the tester can work within that range and avoid wasting time evaluating products that are too expensive.

Expectations and assignments differ depending on the task at hand and the skill set of the tester. Different types of tests, test approaches, techniques, and outcomes may be expected.

Once expectations are set, the test manager can start comparing the work of the test team against the established goals, tasks, and schedules to measure effectiveness of implementation. Following is a list of points to consider when evaluating a tester's effectiveness.

- *Subject-matter expert vs. technical expert.* The expertise expected from a subject-matter expert is related to the domain of the application, while a technical tester is concerned with the technical issues of the application.

  When a technical tester functions as an automater, automated test procedures should be evaluated based on defined standards that must be followed by the test engineers. For example, the supervisor might ask: Did the engineer create maintainable, modular, reusable automated scripts, or do the scripts have to be modified with each new system build? Did the tester follow best practices, such as making sure the test database was baselined and could be restored when the automated scripts need to be rerun? If the tester is developing custom test scripts or a test harness, the tester will be evaluated on some of the same criteria as a developer, including readability and reliability of the code.

  A tester who specializes in the use of automated tools, yet does not understand the intricacies of the application's functionality and underlying concepts, will usually be ineffective. Automated scripts based only on high-level knowledge of the application will often find less-important defects. It is important that the automater understand the application's functionality in order to be an effective member of the testing team.

  Another area for evaluation is technical ability and adaptability. Is the test engineer capable of picking up new tools and becoming familiar with their

capabilities? Testers should be trained regarding the various capabilities of a testing tool, if they are not already thoroughly familiar with them.

- *Experienced vs. novice tester.* As previously mentioned, the skill level of the tester must be taken into account. For example, novice testers may overlook some errors, or not realize they are defects. It is important to assign novice testers to lower-risk testing areas.

  Inexperienced testers are not alone in overlooking defects. Experienced testers may ignore some classes of defects based on past experience ("the product has always done that") or the presence of work-arounds. Appropriately or not, testers may become "acclimated" to familiar errors, and may not report defects that seem unimportant to them but may be unacceptable to end users.

- *Functional vs. nonfunctional testing.* A tester's understanding of the various testing techniques available (see Chapter 5) and knowledge of which technique is most effective for the task at hand should be evaluated. If the tester doesn't understand the various techniques and applies a technique inappropriately, test designs, test cases, and test procedures will be adversely affected.

  Functional testing can additionally be based on a review of the test procedures. Typically, testers are assigned to test procedures for testing specific areas of functionality based on assigned requirements. Test procedure walk-throughs and inspections should be conducted that include the requirements, testing, and development teams. During the walk-through, it should be verified that all teams agree on the behavior of the application.

  The following questions should be considered during an evaluation of functional test procedures:

  - How completely are the test-procedure steps mapped to the requirements steps? Is traceability complete?
  - Are the test input, steps, and output (expected result) correct?
  - Are major testing steps omitted in the functional flow of the test procedure?
  - Has an analytical thought process been applied to produce effective test scenarios?
  - Have the test-procedure creation standards been followed?

- How many revisions have been required as a result of misunderstanding or miscommunication before the test procedures could be considered effective and complete?
- Have effective testing techniques been used to derive the appropriate set of test cases?

During a test-procedure walk-through, the "depth" or thoroughness of the test procedure should be verified. In other words, what does the test procedure test? Does it test functionality only at a high level, or does it really dig deep down into the underlying functionality of the application?

To some extent, this is related to the depth of the requirement steps. For example, a functional requirement might state, "The system should allow for adding records of type A." A high-level test procedure establishes that the record can be added through the GUI. A more-effective test procedure also includes steps that test the areas of the application affected when this record is added. For instance, a SQL statement might verify that the record appears correctly in the database tables. Additional steps could verify the record type. There are numerous other testing steps to be considered, such as verifying the system's behavior when adding multiple records of type A—whether duplicates are allowed, for example.

If test procedures are at a very high level, it is important to confirm that the requirements are at the appropriate level and pertinent details are not missing. If there is a detail in the requirement that is missing in the test procedure, the test engineer might need coaching on how to write effective test procedures. Or, it could be that the engineer did not adequately understand the requirement.

Different criteria apply to evaluating functional testing than to nonfunctional testing. For example, nonfunctional tests must be designed and documented in a different manner than functional test procedures.

- *Testing phase.* Different tasks are to be performed by the tester depending on the testing phase (alpha test, beta test, system test, acceptance test, and so on).

During system testing, the tester is responsible for all testing tasks described in this book, including the development and execution of test procedures, tracking defects to closure, and so on. Other testing phases may be less comprehensive.

During alpha testing, for example, a tester might be tasked with simply recreating and documenting defects reported by members of a separate "alpha testing team," which is usually the company's independent testing (Independent Verification and Validation, or IV&V) team.

During beta testing, a tester might be tasked with documenting the beta-test procedures to be executed, in addition to recreating and documenting defects found by other beta testers. (Customers are often recruited to become beta testers.)

- *Phase of the development life cycle.* As mentioned throughout this book, testers should be involved from the beginning of the life cycle. Evaluation of tester performance should be appropriate to each phase. For example, during the requirements phase, the tester can be evaluated based on defect-prevention efforts, such as identification of testability issues or requirements inconsistencies.

  While a tester's evaluation can be subjective, many variables related to the phase of testing must be considered, rather than jumping to the first seemingly obvious conclusion. For example, when evaluating the test engineer during the requirements phase, it is important to consider the quality of the requirements themselves. If the requirements are poorly written, even an average tester can find many defects. However, if the requirements are well laid out and their quality is above average, only an exceptional tester is likely to find the most subtle defects.

- *Following of instructions and attention to detail.* It is important to consider how well a test engineer follows instructions and pays attention to detail. Reliability and follow-through must be monitored. If test procedures must be updated and executed to ensure a quality product, the test manager must be confident that the test engineers will carry out this task. If tests have to be automated, the test manager should be confident that progress is being made.

  Weekly status meetings where engineers report on their progress are useful to track and measure progress. In the final stages of a testing phase, these meetings may be held daily.

- *Types of defects, defect ratio, and defect documentation.* The types of defects found by the engineer must be considered during the evaluation. When using this metric to evaluate a tester's effectiveness, some factors to keep in mind include the skill level of the tester, the types of tests being performed, the testing phase being conducted, and the complexity and the maturity of

the application under test. Finding defects depends not only upon the skill of the tester, but also on the skill of the developer who wrote, debugged, and unit tested the code, and on the walk-through and inspection teams that reviewed the requirements, design, and code. Ideally, they will have corrected most defects before formal testing.

An additional factor to evaluate in this context is whether the test engineer finds errors that are complex and domain related, or only cosmetic. Cosmetic defects, such as missing window text or control placement, are relatively easy to detect and become high priority during usability testing, whereas more complicated problems relating to data or cause-effect relationships between elements in the application are more difficult to detect, require a better understanding of the application, and become high priority during functional testing. On the other hand, cosmetic-defect fixes, since they are most visible, may have a more immediate effect on customer happiness.

The test manager must consider the area for which the tester is responsible. The tester responsible for a specific area where the most defects are discovered in production should not necessarily be assumed to have performed poorly. If the tester's area is very complex and error-prone and the product was released in a hurry, failure to catch some defects may be understandable.

The *types* of defects discovered in production also matter. If they could have been discovered by a basic test within the existing test-procedure suite, and if there was plenty of time to execute the test procedure, this would be a major oversight by the tester responsible for this area. However, before passing judgment, some additional questions should be considered:

- Was the test procedure supposed to be executed manually? The manual tester may have become tired of executing the same test procedures over and over, and after many trials concluded it should be safe not to execute the tests because that part of the application has always worked in the past.
- Was the software delivered under pressure of a deadline that could not be changed even though it ruled out a full test cycle? Releases should not be allowed without having met the release criteria, time pressures notwithstanding.

- Was this test automated? Did the automated script miss testing the step containing the error? In such a case, the automated scripts must be reevaluated.
- Was the defect discovered using some combination of functional steps that are rarely executed? This type of defect is more understandable.

Additionally, it may be necessary to review the test goals, risks of the project, and assumptions made when the test effort started. If it had been decided not to conduct a specific type of test because of time constraints or low risk, then the tester should not be held responsible. This risk should have been taken with full knowledge of the possibility of problems.

Effectiveness can also be evaluated by examining how a defect is documented. Is there enough detail in the documented defect for a developer to be able to recreate the problem, or do developers have a difficult time recreating one specific tester's defects? Standards must be in place that document precisely what information is required in defect documentation, and the defect tracking life cycle must be well communicated and understood. All testers must follow these standards. (For a discussion of the defect tracking life cycle, see Item 50.)

For each issue uncovered during evaluation of a tester, the cause of the issue should be determined and a solution should be sought. Each issue must be evaluated with care before a judgment regarding the tester's capability is made. After careful evaluation of the entire situation, and after additional coaching has been provided where called for, it will be possible to evaluate how detail oriented, analytical, and effective this tester is. If it is determined that the tester lacks attention to detail or analytical skills or there are communication issues, that tester's performance may need to be closely monitored and reviewed, and there may be a need for additional instruction and training, or other appropriate steps need to be taken.

Testers' effectiveness must be constantly evaluated to ensure the success of the testing program.

TEST ENGINEER SELF-EVALUATION

Test engineers should assume responsibility for evaluating their own effectiveness. The following list of issues can be used as a starting-point in developing a process for test-engineer self-evaluation, assuming roles and responsibilities along with task assignments are understood:

- Consider the types of defects being discovered. Are they important, or are they mostly cosmetic, low-priority defects? If the tester consistently uncovers only low-priority defects—such as, during functional testing, non-working hot keys, or typographical errors in the GUI—the effectiveness of the test procedures should be reassessed. Keep in mind that during other testing phases (for the examples mentioned here, during usability testing), the priority of certain defects will change.
- Are test procedures detailed enough, covering the depth, and combinations and variations of data and functional paths, necessary to catch the higher-priority defects? Do tests include invalid data as well as valid data?
- Was feedback regarding test procedures received and incorporated from requirements and development staff, and from other testers? If not, the test engineer should ask for test-procedure reviews, inspections, and walk-throughs involving those teams.
- Does the test engineer understand the range of testing techniques available, such as boundary-values testing, equivalence partitioning, and orthogonal arrays, well enough to select the most effective test procedures?
- Does the engineer understand the intricacies of the application's functionality and domain well? If not, the tester should ask for an overview or additional training. A technical tester may ask for help from a Subject-Matter Expert (SME).
- Are major defects being discovered too late in the testing cycle? If this occurs regularly, the following points should be considered:

  Does the initial testing focus on low-priority requirements? Initial testing should focus on the high-priority, highest-risk requirements.

  Does the initial testing focus on regression testing of existing functionality that was working previously and rarely broke in the past? Initial testing should focus on code changes, defect fixes, and new functionality. Regression testing should come later. Ideally, the regression-testing efforts can be automated, so test engineers can focus on the newer areas.
- Are any areas under testing exhibiting suspiciously low defect counts? If so, these areas should be re-evaluated to determine:

  Whether the test coverage is sufficiently robust.

Whether the types of tests being performed are most effective. Are important steps missing?

Whether the application area under test has low complexity, so that indeed it may be error-free.

Whether the functionality was implemented in such a manner that it is likely no major defects remain—for example, if it was coded by the most senior developers and has already been unit and integration tested well.

Consider the Defect Workflow:

- Each defect should be documented in a timely manner (i.e., as soon as it is discovered and verified)
- Defect-documentation standards must be followed. If there aren't any defect-documentation standards, they should be requested from the engineer's manager. The standards should list all information that must be included in documenting a defect to enable the developer to reproduce it.
- If a new build is received, initial testing should focus on retesting the defects. It is important that supposedly fixed defects be retested as soon as possible, so the developers know whether their repair efforts are successful.
- Comments received from the development team regarding the quality of defect reports should be continually evaluated. If the reports are often said to lack required information, such as a full description of testing steps required to reproduce the errors, the testers should work on providing better defect documentation.
- Testers should be eager to track defects to closure.

- Examine the comments added to defect documentation to determine how developers or other testers receive it. If defects are often marked "works as expected" or "cannot be reproduced," it could also signal some problems:
  - The tester's understanding of the application may be inadequate. In this case, more training is required. Help may be requested from domain SMEs.
  - The requirements may be ambiguous. If so, they must be clarified. (Most commonly, this is discovered during the requirements or test-procedure walk-through and inspections.)

- The engineer's documentation skills may not be as effective as necessary. Inadequate documentation may lead to misunderstanding of the identified defect. The description may need additional steps to enable developers to reproduce the error.
- The developer may be misinterpreting the requirement.
- The developer may lack the patience to follow the detailed documented defect steps to reproduce the defect.

- The tester should monitor whether defects are being discovered in that person's test area after the application has gone to production. Any such defects should be evaluated to determined why they were missed:
  - Did the tester fail to execute a specific test procedure that would have caught this defect? If so, why was the procedure overlooked? Are regression tests automated?
  - Was there no test procedure that would have caught this defect? If so, why not? Was this area considered low risk? The test procedure creation strategy should be reevaluated, and a test procedure should be added to the regression test suite to catch errors like the one in question. The tester should discuss with peers or a manager how to create more effective test procedures, including test design, strategy, and technique.
  - Was there not enough time to execute an existing test procedure? If so, management should be informed—before the application goes live or is shipped, not after the fact. This sort of situation should also be discussed in a post-test/pre-installation meeting, and should be documented in the test report.
- Do other testers during the course of their work discover defects that were this tester's responsibility? If so, the tester should evaluate the reasons and make adjustments accordingly.

There are many more questions a tester can ask related to testing effectiveness, depending on the testing phase and testing task at hand, type of expertise (technical vs. domain), and tester's experience level.

An automater might want to be sure to become familiar with automation standards and best automation practices. A performance tester might request additional training in the performance-testing tool used and performance testing techniques available.

Self-assessment of the tester's capabilities and the improvement steps that follow are important parts of an effective testing program.