**Chapter 8. Software Testing**

**The quality of the software is the responsibility of the entire team**. Throughout the entire software project, the team does many things to find and prevent defects. Once the software has been built, it's time to look back and make sure that it meets the requirements. The goal of software testing is to make sure that the product does what the users and stakeholders need it to do. Software testers review the final product to make sure that the initial requirements have been met.

In software testing, *quality* is defined as "conformance to requirements." Every use case, functional requirement, and other software requirement defines a specific behavior that the software must exhibit. When the software does not behave the way that the requirements say it must behave, that is a *defect*. This means that your software testers are responsible for figuring out whether the software that was produced by the team behaves in the way that the requirements it was built from say that it should.

Every engineering discipline defines quality in exactly this way. When an auto manufacturer provides a specification to a subcontractor to create a part for a car, that specification contains tolerances for various measurements for that part. When the subcontractor ships a box of those parts back to the manufacturer to be built into the car, any part that does not meet the requirements in the specification is considered defective. And it's exactly the same with software—any feature that does not meet the requirements in the specification has a defect.

This means that if an organization does not have good requirements engineering practices (see [Chapter 6](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch06.html)), then it will be very hard to deliver software that fills the users' needs, because the product team does not really know what those needs are. It's not a coincidence that many problems that seem to originate with the software testers are really requirements problems that simply have not been caught yet.

This chapter gives an overview of what testers do in a software organization, and clears up some misconceptions about what they do not do. By putting in place good software testing practices based on solid software requirements specifications, a project manager can help assure the quality of the software.

**Test Plans and Test Cases**

The goal of test planning is to establish the list of tasks that, if performed, will identify all of the requirements that have not been met in the software. The main work product is the *test plan*. There are many standards that can be used for developing test plans . [Table 8-1](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch08.html" \l "appliedprojectmgmt-CHP-8-TABLE-1) shows the outline of a typical test plan. (This outline was adapted from IEEE 829, the most common standard for software test plans.)

*Table 8-1. Test plan outline*

|  |
| --- |
| *Purpose*  A description of the purpose of the application under test.  *Features to be tested*  A list of the features in the software that will be tested. It is a catalog of all of the test cases (including a test case number and title) that will be conducted, as well as all of the base states.  *Features not to be tested*  A list of any areas of the software that will be excluded from the test, as well as any test cases that were written but will not be run.  *Approach*  A description of the strategies that will be used to perform the test.  *Suspension criteria and resumption requirements*  Suspension criteria are the conditions that, if satisfied, require that the test be halted. Resumption requirements are the conditions that are required in order to restart a suspended test.  *Environmental Needs*  A complete description of the test environment or environments. This should include a description of hardware, networking, databases, software, operating systems, and any other attribute of the environment that could affect the test.  *Schedule*  An estimated schedule for performing the test. This should include milestones with specific dates.  *Acceptance criteria*  Any objective quality standards that the software must meet, in order to be considered ready for release. This may include things like stakeholder sign-off and consensus, requirements that the software must have been tested under certain environments, minimum defect counts at various priority and severity levels, minimum test coverage numbers, etc.  *Roles and responsibilities*  A list of the specific roles that will be required for people in the organization, in order to carry out the test. This list can indicate specific people who will be testing the software and what they are responsible for. |

The test plan represents the overall approach to the test. In many ways, the test plan serves as a summary of the test activities that will be performed. It shows how the tests will be organized, and outlines all of the testers' needs that must be met in order to properly carry out the test. The test plan is especially valuable because it is not a difficult document to review, so the members of the engineering team and senior managers can inspect it.

The bulk of the test planning effort is focused on creating the *test cases*. A test case is a description of a specific interaction that a tester will have, in order to test a single behavior of the software. Test cases are very similar to use cases, in that they are step-by-step narratives that define a specific interaction between the user and the software. However, unlike use cases, they contain references to specific features of the user interface. The test case contains actual data that must be entered into the software and the expected result that the software must generate. A typical test case includes these sections, usually laid out in a table:

* A unique *name* and *number*
* A *requirement* that this test case is exercising
* *Preconditions* that describe the state of the software before the test case (which is often a previous test case that must always be run before the current test case)
* *Steps* that describe the specific steps that make up the interaction
* *Expected results* that describe the expected state of the software after the test case is executed

[Table 8-2](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch08.html#appliedprojectmgmt-CHP-8-TABLE-2) shows an example of a test case that would exercise one specific behavior in requirement FR-4 from the discussion of functional requirements in [Chapter 6](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch06.html). This requirement specified how a search-and-replace function must deal with case sensitivity. One part of that requirement said, "If the original text was all lowercase, then the replacement text must be inserted in all lowercase."

*Table 8-2. Example of a test case*

|  |  |
| --- | --- |
| Name | TC-47: Verify that lowercase data entry results in lowercase insert |
| Requirement | FR-4 (Case sensitivity in search-and-replace), bullet 2 |
| Preconditions | The test document TESTDOC.DOC is loaded (base state BS-12). |
| Steps | 1. Click on the "Search and Replace" button. 2. Click in the "Search Term" field. 3. Enter *This is the Search Term*. 4. Click in the "Replacement Text" field. 5. Enter *This IS THE Replacement TeRM*. 6. Verify that the "Case Sensitivity" checkbox is unchecked. 7. Click the OK button. |
| Expected results | 1. The search-and-replace window is dismissed. 2. Verify that in line 38 of the document, the text *this is the search term* has been replaced by *this is the replacement term*. 3. Return to base state BS-12. |

Note that this test case includes interactions with design elements like text fields, buttons, and windows. This is one of the main differences between a use case and a test case. A use case specifically does not talk about design elements, in order to avoid constraining the designers. Test cases must be very specific about how they plan on doing their interaction because the design has been decided upon, and the part of the purpose of the test case is to exercise that design. This means that the test case cannot be completed until the design of the software is finished.

A project manager should be aware of the characteristics of a good test case. A test case describes in complete detail everything that the tester must do. It contains the names of the buttons that the tester must click on, the menu items that should be selected, the exact data that must be typed in, etc. All of the expected behavior of the software (such as windows dismissed or error messages displayed) must be described. The goal of that is to make each test case repeatable, so that no two people will test the software differently.

[Table 8-3](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch08.html#appliedprojectmgmt-CHP-8-TABLE-3) shows an example of a test case that is widely open to interpretation. It may seem specific at first glance, but there are some serious problems with it:

* The test case does not specify exactly how the search-and-replace function is accessed. If there are several ways to bring up this function, and a defect is found that is specific to only one of them, it may be difficult to repeat precisely.
* The test case is not data-specific. Every tester could enter a different search term and replacement term. If a defect only occurs for certain terms, it will be difficult to reproduce.
* The test case does not specify how the data is entered into the field. It is possible that a problem might come up when the user uses the tab key to navigate between fields, but cannot be reproduced by clicking on them.

*Table 8-3. This poorly designed test case does not describe the interaction precisely enough*

|  |  |
| --- | --- |
| Steps | 1. Bring up search-and-replace. 2. Enter a lowercase word from the document in the search term field. 3. Enter a mixed-case word in the replacement field. 4. Verify that case sensitivity is not turned on and execute the search. |
| Expected results | 1. Verify that the lowercase word has been replaced with the mixed-case term in lowercase. |

In short, this test case is not repeatable. It may seem intuitive to make the test case more general, in order to capture a wider range of functionality. However, the test case itself will be run only once during each test iteration. Instead of trying to make the test case more general, multiple test cases should be added to the test plan, in order to verify each specific type of test. For example, there should be separate test cases for clicking in the text field and tabbing between them. If the tester wants to verify that the find-and-replace function works with long strings as well as short ones, or for numbers and symbols as well as alphabetical characters, all of those things should be separate test cases. (The test case name should be used to differentiate between these tests.)

Another important characteristic is that each test case describes one—and only one—test. The reason for this is that the test case should isolate the specific behavior that is being tested, to ensure that any defect that is found is a problem with that feature only. One of the complexities of software is that there are usually an infinite number of possible feature combinations, and the only way to make sure that those combinations are not interacting improperly is to isolate each specific behavior. That makes it much easier to determine the root cause of any defect found.

For example, [Table 8-4](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch08.html#appliedprojectmgmt-CHP-8-TABLE-4) contains the "Expected Results" section of a poorly designed test case that exercises all of the bullet points in requirement FR-4. If a defect were found in bullet point number 4, it would be difficult to determine whether the defect arose because those specific actions were done in sequence, or if it were simply an isolated defect.

*Table 8-4. This poorly designed test case has more than one interaction*

|  |  |
| --- | --- |
| Expected Results | 1. The search and replace window is dismissed. 2. Verify that in line 36 of the document, the text *THIS IS THE SEARCH TERM* has been replaced by *THIS IS THE REPLACEMENT TERM*. 3. Verify that in line 38 of the document, the text *this is the search term* has been replaced by *this is the replacement term*. 4. Verify that in line 43 of the document, the text *This is the search term* has been replaced by *This is the replacement term*. 5. Verify that in line 44 of the document, the text *This Is the Search Term* has been replaced by *This Is the Replacement Term*. 6. Verify that in line 44 of the document, the text *thIS is the SEarCh Term* has been replaced by *This IS THE Replacement TeRM*. |

Test cases are usually strung together in one long interaction with the software. This means that the results in [Table 8-4](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch08.html#appliedprojectmgmt-CHP-8-TABLE-4) should really be verified using five different test cases, one per bullet point in requirement FR-4. For example, since test case TC-47 verifies the second bullet point in FR-4, TC-48 could verify the third bullet point and have in its precondition that TC-47 has been run. The precondition for TC-47, in turn, would require that TC-46 be run.

To ensure that the test cases all start out with the same document open, each test case depends on a *base state* , or a condition of the software that can be reproduced at any time. A base state is an anchor point that is easy to navigate to. Test case TC-47 contains two references to a base state labeled BS-12. The first reference is in the Precondition section: the test case requires that the software be in its base state. The second reference is at the end of the Expected Results section: the tester must return the software to the base state after the test case, in order to reset it for the next one. This ensures that whether the test passes or fails, it will not have any side effects on any tests that are executed after it. [Table 8-5](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch08.html#appliedprojectmgmt-CHP-8-TABLE-5) shows the definition of this base state. It is in the same form as a test case (note that since it is not exercising a particular requirement, the "Requirement" section contains the text "N/A").

*Table 8-5. Base state BS-12*

| **Name** | **BS-12: Load test document TESTDOC.DOC** |
| --- | --- |
| Requirement | N/A |
| Preconditions | No user is logged in and no applications are running. |
| Steps | 1. Log in as user "joetester" with password "test1234". 2. Launch the application. 3. Select the File/Open menu item. 4. Enter "/usr/home/joetester/TESTDOC.DOC". 5. Click OK. |
| Expected Results | 1. Verify that the Open Window dialog box has been dismissed. 2. Verify that the file TESTDOC.DOC is loaded. 3. Verify that the file TESTDOC.DOC is given focus. 4. Verify that the file TESTDOC.DOC is active. |

It is not necessary for every test case to start out in a base state. In fact, it is often useful to string a set of test cases together so that the precondition of each one depends on the previous test case passing. However, when there are strings of test cases that go for a long time without returning to a base state, there is a risk that areas of the application will go untested in the event of a failure. If a test case fails, the results of the following test cases simply cannot be trusted until the software is returned to a base state.

Once all of the test cases and base states are defined, they should be combined into a single *test case document* . This document is usually far longer than any other document produced over the course of the software project. It contains a separate table for each test case and base state. Each of the test cases and base states should be cross-referenced with the "Features to be Tested" section of the test plan . This section should contain the complete Name field of each test case and base state. Typically, the test cases and base states appear in the test case document in the same order that they appear in the test plan.

The test case document should have an outline that follows the software requirements specification: it should contain one section for each use case and requirement, and in that section, there should be a set of test cases that fully test that requirement. This makes the test cases much easier to inspect, because a reviewer can look at a single section and judge whether the test cases in that section fully exercise the requirement that they are supposed to be testing.

Once the test cases are complete, they should be inspected by the engineering team (see [Chapter 5](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch05.html)). The test plan and test cases should be collaborative documents; if the team does not give input into them, then it is likely that the software will fail to implement certain behavior that the users expect. This inspection will generally have a narrower audience than the test plan because the document is much longer and more technical. Minimally, it should be reviewed by another software tester, the requirements engineer who built the requirements that are being tested, and the programmer who implemented them.

**Inspection Checklist**

The following checklist items apply to the test plan.

*Completeness*

Does the document meet all established templates and standards?

Is the document complete?

Are there any requirements that are not tested?

Are there any features that are planned for testing but should be excluded?

*Feasibility*

Can the testing as planned be accomplished within the known cost and schedule constraints?

Can every test described in the test plan be reasonably conducted?

*Environment*

Is the description of the environment complete?

Is the test plan traceable to any nonfunctional requirements that define the operating environment?

*Performance*

Does the test plan account for the expected load for concurrent users, large databases, or other performance requirements?

Can the performance tests be traced back to requirements in the specification?

*Acceptance Criteria*

Do the acceptance criteria match the standards of the organization?

The following checklist items apply to the test cases:

*Clarity*

Does each test case have a clear flow of events?

Does each test case test only one specific interaction?

Does each test case describe the interaction using specific user interface and data elements?

Is each test case repeatable by someone uninitiated on the project?

*Completeness*

Is every requirement in the SRS verified fully with individual test cases?

Are all of the steps in each test case necessary?

Are there any steps that are missing?

Are all alternative paths and exceptions accounted for?

*Accuracy*

For every action, is there an expected result?

For every behavior in the requirement, is there a verification of the actual behavior?

Is the test case data specific—if data must be entered or modified, is that data provided?

*Traceability*

Is each test case uniquely identified with a name and a number?

Can each test case be traced back to a specific requirement?

**Test Execution**

The software testers begin executing the test plan after the programmers deliver a build that they feel is feature complete. This is referred to as the *alpha build* . The alpha should be of high quality—the programmers should feel that it is ready for release, and as good as they can get it. This build should have been code reviewed (see [Chapter 5](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch05.html)) and should have passed unit tests (see [Chapter 7](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch07.html)); it should have already been minimally functionally tested by the development team, as well.

There are typically several iterations of test execution . The first iteration focuses on new functionality that has been added since the last round of testing. (If this is the first time this software product has been tested, then every test case is executed.) If no defects are uncovered that are considered high enough priority to fix (see below about defect triage), then the testers move on to perform a regression test .

A *regression test* is a test designed to make sure that a change to one area of the software has not caused any other part of the software that had previously passed its tests to stop working. Regression testing usually involves executing all test cases that have previously been executed. In other words, it's not enough to verify that the software has been altered: it's also necessary to ensure that the change did not break any other part of the software that previously worked.

It is rare for no defects to be uncovered in the first test iteration. Usually, some test cases fail and defects must be reported. Once the iteration is complete, the defects are triaged (see below) and the programmers begin repairing the software. When a new build is delivered, the next iteration of testing begins.

After each iteration, the testers create a *test report*. This is a document (usually a spreadsheet or word processor document) that simply contains a list of all test cases that failed or were not executed. The purpose of the report is to give the project team a good idea of how much of the application was exercised in the test.

For each test case that failed, a tester creates a *defect report*. These reports are used to determine the general health of the product, in order to allow the project team and stakeholders to understand its maturity and readiness for release (see below about defect tracking).

Testing is complete when either no defects are found or (more likely) all of the defects that have been found satisfy the acceptance criteria in the test plan. These criteria will typically include several rules: that all test cases have been executed, that the project stakeholders have reviewed all of the defects and personally determined that the software can be released with those known defects, and that there is consensus among the engineering team that the product is ready for release. [Table 8-6](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch08s02.html#appliedprojectmgmt-CHP-8-TABLE-6) shows an example of typical acceptance criteria from a test plan.

*Table 8-6. Acceptance criteria from a test plan*

|  |
| --- |
| 1. Successful completion of all tasks as documented in the test schedule. 2. Quantity of medium- and low-level defects must be at an acceptable level as determined by the software testing project team lead. 3. User interfaces for all features are functionally complete. 4. Installation documentation and scripts are complete and tested. 5. Development code reviews are complete and all issues addressed. All high-priority issues have been resolved. 6. All outstanding issues pertinent to this release are resolved and closed. 7. All current code must be under source control, must build cleanly, the build process must be automated, and the software components must be labeled with correct version numbers in the version control system. 8. All high-priority defects are corrected and fully tested prior to release. 9. All defects that have not been fixed before release have been reviewed by project stakeholders to confirm that they are acceptable. 10. The end user experience is at an agreed acceptable level. 11. Operational procedures have been written for installation, set up, error recovery, and escalation. 12. There must be no adverse effects on already deployed systems. |

The list of acceptance criteria can also include any specific performance requirements ("server must support 80 users," "product must perform under high load for 72 hours without failure," etc.), as well as any security requirements specific to the software.

The majority of the acceptance criteria in the example focus on the consensus of the people involved in the software project. It is important that everyone agrees that any defects that the software ships with will not affect the users.

Note that these criteria assume that there will be defects that will not be fixed. The goal of test execution is not to remove every defect from the software; rather, it is to make the people in the organization aware of the individual issues encountered during testing. This way, everyone is aware of the risks (if any) involved in releasing the software, so an informed decision can be made about whether or not to fix the defects and retest the software.