Table 6.8. Testing objectives and activities during operation and maintenance

Objectives	Activities
Efficient regression testing	Capture user problems Perform regression testing

updated software still possesses the functionality it had before the updates, as well as the new or modified functionality. Table 6.8 summarizes the major objectives and activities during operation and maintenance.

6.3.9 Summary

A key factor to instilling quality into a development process is based on individual professional ethics. Developers and testers alike can choose to **put quality first**. If the process is such that the tester does not know how to test it, then don't build it. This will sometimes result in conflicts with time-driven management, but even if you lose the argument, you will gain respect. It is important that developers begin test activities early. It also helps to take a stand against taking shortcuts. Almost all projects will eventually be faced with taking shortcuts that will ultimately reduce the quality of the software. Fight it! If you lose the argument you will gain respect: document your objections, vote with your feet, and don't be afraid to be right!

It is also essential that test artifacts be managed. A lack of organization is a sure recipe for failure. Put test artifacts under version control, make them easily available, and update them regularly. These artifacts include test design documents, tests, test results, and automated support. It is important to keep track of the criteria-based source of the tests, so when the source changes, it is possible to track which tests need to change.

6.4 TEST PLANS

A major emphasis for many organizations is documentation, including test plans and test plan reporting. Unfortunately, putting too much of a focus on documentation can lead to an environment where lots of meaningless reports are produced but nothing useful is done. That is why this book focuses on content, not form. The contents of a test plan are essentially how the tests were created, why the tests were created, and how they will be run.

Producing test plans, however, is an essential requirement for many organizations. Companies and customers often impose templates or outlines. Rather than surveying many different types of test plans, we look at the IEEE standard definition. Unfortunately, this is quite old (1983!), but it is still the most widely known. A quick search on the Web will supply you with more test plans and test plan outlines than you could ever use. ANSI/IEEE Standard 829-1983 describes a test plan as:

"A document describing the scope, approach, resources, and schedule of intended testing activities. It identifies test items, the features to be tested, the testing tasks, who will do each task, and any risks requiring contingency planning."

Several different general types of test plans are:

1. A *mission plan* tells "why." Usually only one mission plan appears per organization or group. Mission plans describe the reason that the organization exists, are typically very short (5–10 pages), and are the least detailed of all plans.
2. A *strategic plan* tells "what" and "when." Again, only one strategic plan usually is used per organization, although some organizations develop a strategic plan for each category of project. Strategic plans can say things such as "we will always do Edge Coverage during unit testing" and "Integration Testing will be driven by couplings." Strategic plans are more detailed and longer than mission plans, sometimes 20–50 pages or more. They are seldom detailed enough to be directly useful to practicing testers or developers.
3. A *tactical plan* tells "how" and "who." Most organizations use one overall tactical plan per product. Tactical plans are the most detailed of all plans, and are usually living documents. That is, a tactical plan may begin life as a table of contents, and be continually added to during the life of the product or product development. For example, a tactical test plan would specify how each individual unit will be tested.

Below are outlines of two sample test plans, provided as example only. The plans were derived from numerous samples that have been posted on the Web, so do not exactly represent a single organization. The first is for system testing and the second is a tactical plan for unit testing. Both are based on the IEEE 829-1983 standard.

1. Purpose
   The purpose of a **test plan** is to define the strategies, scope of testing, philosophy, test exit and entrance criteria, and test tools that will be used. The plan should also include management information such as resource allocations, staff assignments, and schedules.
2. Target Audience and Application
   (a) The test staff and quality assurance personnel must be able to understand and implement the test plan.
   (b) The quality assurance personnel must be able to analyze the results and make recommendations on the quality of the software under test to management.
   (c) The developers must be able to understand what functionalities will be tested and the conditions under which the tests are to be performed.
   (d) The marketing personnel must be able to understand with which configurations (hardware and software) the product was tested.
   (e) Managers must understand the schedule to the degree of when testing is to be performed and when it will be finished.
3. Deliverables
   The results of testing are the following deliverables:
   (a) Test cases, including input values and expected results
   (b) Test criteria satisfied
   (c) Problem reports (generated as a result of testing)
   (d) Test coverage analysis

4. Information Included
   Each test plan should contain the following information. Note that this can (and often does) serve as the outline of the actual test plan, and can be tailored to most environments successfully.
   (a) Introduction
   (b) Test items
   (c) Features tested
   (d) Features not tested (per cycle)
   (e) Test criteria, strategy and approach
      - Syntax
      - Description of functionality
      - Argument values for tests
      - Expected output
      - Specific exclusions
      - Dependencies
      - Test case success criteria
   (f) Pass/fail standards
   (g) Criteria for beginning testing
   (h) Criteria for suspending test and requirements for restarting
   (i) Test deliverables/status communications documents
   (j) Hardware and software requirements
   (k) Responsibilities for determining problem severity and correcting problems
   (l) Staffing and training needs
   (m) Test schedules
   (n) Risks and contingencies
   (o) Approvals

The above plan is in a very general and high level style. The next example is in a much more detailed style, and is more suited for tactical test plans for engineers.

1. Purpose
   The purpose of the test plan is to describe the scope, approach, resources, and schedule of all testing activities. The plan should identify the items to be tested, the features to be tested, the testing tasks to be performed, the personnel responsible for each task, and the risks associated with this plan.

   The test plan should be a dynamic document that can be used by testers, managers, and developers. The test plan should evolve as the project evolves. At the end of the project the test plan should document the activities and be the vehicle by which all parties sign indicating approval of the final product.
2. Outline
   A test plan has the following structure:
   (a) Test-plan identifier
   (b) Introduction
   (c) Test reference items
   (d) Features that will be tested

(e) Features that will not be tested
(f) Approach to testing
(g) Criteria for pass/fail
(h) Criteria for suspending test and requirements for restarting
(i) Test deliverables
(j) Testing tasks
(k) Environmental needs
(l) Responsibilities
(m) Staffing and training needs
(n) Schedule
(o) Risks and contingencies
(p) Approvals

The sections are ordered in the sequence above. Additional sections may be included if necessary. If some or all of the content of a section is in another document, a reference to that document can be listed. The referenced material must be easily available. The following sections give details on the content of each section.

3. Test-Plan Identifier
   Give a unique identifier (name) to this test plan.
4. Introduction
   Give a description or purpose of the software, so that both the tester and the client are clear as to the purpose of the software and the approach to be taken in testing.
5. Test Reference Items
   Identify items that are referred to by the tests, including their version/revision and dates. Supply references to the following documents, if available:
   (a) Requirements specification
   (b) Design specification
   (c) Users guide
   (d) Operations guide
   (e) Installation guide
   (f) Analysis diagrams, including data flow, etc.
   (g) UML or other modeling documents
6. Features that Will Be Tested
   Identify all features and feature combinations that need to be tested. Identify the test design that is associated with each feature and each combination of features.
7. Features that Will not Be Tested
   Identify all features and significant combinations of features that will not be tested. Most importantly, state why.
8. Approach to Testing
   For each major group of features or feature combinations, specify the approach that will ensure that these feature groups are adequately tested. Specify the major activities, criteria, and tools that will be used.

   The approach should be described in enough detail to identify the major testing tasks and estimate how long each will take.

9. Criteria for Pass/Fail
   Specify the measure to be used to determine whether each test item has passed or failed testing. Will it be based on a criterion? The number of known faults?
10. Criteria for Suspending Testing and Requirements for Restarting
    In certain situations, testing must stop and the software sent back to the developers. Specify the criteria used to suspend all or any portion of the testing. Specify the activities that must be repeated to resume or restart testing activities.
11. Test Deliverables
    Identify the documents that should be included in the report. The following are candidate documents.
    (a) Test plan
    (b) Test design specifications
    (c) Test case specifications
    (d) Test process
    (e) Test logs
    (f) Test trouble reports
    (g) Test summary reports
    (h) Test input data and test output data (or where they are located)
12. Testing Tasks
    Identify the tasks necessary to prepare for and perform testing. Identify all dependencies among the tasks.
13. Environmental Needs
    Specify both the necessary and desired properties of the test environment. This specification should contain:
    (a) The physical characteristics of the facilities, including the hardware
    (b) Any communications and system software
    (c) The mode of usage (stand-alone, transient, web-based, etc.)
    (d) Any other software or supplies needed to run the test
    (e) Test tools needed
    (f) Any other testing needs (e.g., publications) and where to get them
14. Responsibilities
    Identify the groups responsible for all aspects of testing and correcting problems. In addition, identify the groups responsible for providing the test reference items identified and the environmental needs above.
15. Staffing and Training Needs
    Specify test staffing needs in terms of knowledge and skill. Identify training options when appropriate and necessary.
16. Schedule
    Include all test milestones identified in the software project schedule. Define any additional test milestones needed. Estimate the time required to do each testing task and specify the schedule for each testing task and test milestone.
17. Risks and Contingencies
    Identify any risky assumptions of the test plan. For example, specialized

knowledge may be needed but not available. Specify contingency plans for each.

18. Approvals
    Specify the names and titles of all persons who must approve this plan, and include space for them to sign and date the document.

6.5 IDENTIFYING CORRECT OUTPUTS

The main contribution of this book is set of coverage criteria for testing. But no matter what coverage criterion is used, sooner or later one wants to know whether a given program executes correctly on a given input. This is the *oracle* problem in software testing.

The oracle problem can be surprisingly difficult to solve, and so it helps to have a range of approaches available. This section describes several common approaches to the oracle problem.

6.5.1 Direct Verification of Outputs

If you are lucky, your program will come with a specification, and the specification will be clear as to what output accompanies a given input. For example, a *sort* program should produce a permutation of its input in a specified order.

Having a human evaluate the correctness of a given output is often effective, but is also expensive. Naturally, it is cheaper to automate this process. Automating direct verification of the output, when possible, is one of the best methods of checking program behavior. Below is an outline of a checker for sorting. Notice that the checking algorithm is *not* another sorting algorithm. It is not only different, it is not particularly simple. That is, writing output checkers can be hard.

```
Input: Structure S
   Make copy T of S
   Sort S
    // Verify S is a permutation of T
    Check S and T are of the same size
    For each object in S
        Check if object appears in S and T same number of times
    // Verify S is ordered
    For each index i but last in S
        Check if S[i] <= S[i+1]
```

Unfortunately, direct verification is not always possible. Consider a program that analyzes Petri nets, which are useful for modeling processes with state. One output of such analysis is the probability of being in any given state. It is difficult to look at a given probability and assert that it is correct – after all, it is just a number. How do you know if all of the digits are, in fact, the right ones? For Petri nets, the final probabilities cannot easily be related back to the input Petri net.