

# A Study in Test Case Styles

Presented to the Quality Week International 2006 Conference, Toronto, Ontario, by  
Jerrold Landau

Jerrold Landau  
Test and Globalization Coordinator for WebSphere Business Modeler  
IBM Toronto Lab  
8200 Warden Ave.  
Markham, ON  
L6G1C7  
905 413 5874  
landau@ca.ibm.com

Test cases are the fundamental building block of any software test endeavour. However, there is no uniform definition for the concept of a test case. A test case could be a set of English instructions describing how to run a certain piece of program functionality, or a computer program (or subset thereof) that actually runs that functionality. It could be an independent object, or a unit of a larger suite of interrelated test cases. Of course, each test case has to have some basic features – a description of steps to be performed, and a way of determining the result of the test. Aside from those basic features, the test case itself will vary widely in scope, style, and form depending on its purpose and environment. Whatever the style, an effective test case will accurately and concisely describe its steps and result to the user, and leave no room for ambiguity.

In this presentation, we will examine several issues relating to test case style in an attempt to understand how these can enhance or impede a test effort. We will concentrate on manual test cases (that is, test cases that are written as English scripts rather than as computer programs); nevertheless, we will touch on some issues of automated test cases as well.

The style of a set of test cases depends on a number of factors. These factors include the desired efficiency of the test effort, the expectations of the reusability and readability of the test cases, and the linkage of the test cases to a mechanism for recording test results. The preferred style of the test cases varies, and depends on the expectations for each of these factors. For example, a set of tests that is expected to be used once should have a very different style from a set that is expected to be reused with minor modifications for multiple releases of a product. A set of tests that forms the basis of a fully automated run where execution time is not a factor should have a very different style from a set that is expected to be run manually in a repeated fashion. The definition of proper style of a test case does not have a single answer, and will vary based on these factors.

## ***Storage of Test Cases in a Test Management Tool***

Many testing endeavors make use of a test management tool. Each test management tool possesses its own unique features; however, most of provide a framework for test case storage, the linkage of test cases into runnable units, and the storage and reporting of results of both current and historical runs. Some tools provide more advanced features such as a review and approval mechanism, linkage to a defect management system, and room for personal notes and comments regarding test case execution.

At IBM, the test teams have access to a wide variety of test management tools. The choice of tool is often dictated by reasons of history, comfort, and convention. Several years ago, IBM acquired Rational Software Corporation<sup>®</sup>, and integrated the Rational<sup>®</sup> toolset into its product suite. Other home-grown tools include Test Tracking Tool (TTT), TestCase Database, and Universal Verification Test System (UVATS). For historical reasons, the product area that I work with (WebSphere<sup>®</sup> Business Modeler) makes use of TTT.

TTT is an application based on Lotus<sup>®</sup> Notes<sup>®</sup>. Each individual test case is stored in a record that contains the following sections:

Scenario / Approval;  
Configuration / Tools / Flags;  
Procedure / Results;  
History;  
Notes;  
Execution Records;  
User Defined.

A comprehensive review and approval system is included. A mechanism to assign test cases to members of a test team is also included.

Test cases are stored in a two-level hierarchy consisting of functions and categories. Each individual test case can be assigned a weight. Test cases can be grouped together in phases and platforms in order to define a test run. For example, a set of tests can be chosen to be run on the Version 6 phase / Windows<sup>®</sup> 2003 platform.

TTT includes numerous report templates so that test status can be displayed based on any of the grouping areas. For example, a report on test results can be ordered by function and category, by tester, by phase and platform, or by any combination of features that the tool provides. User-defined reports can also be designed for those who want something beyond the wide variety of predefined reports.

TTT is a wonderfully flexible tool. It solves a wide variety of test management issues, including reviews, approvals, selection of subsets of tests for execution, storage of historical data, and reporting of results. However, this range of functionality often comes at the expense of test case readability. If one's concept of a test suite is an orderly description of steps that presents a readily understandable picture of component functionality, TTT will disappoint. Test organization can be maintained reasonably well

when the size of the test suite is restricted to several hundred test cases, but it can become quite cumbersome as the test suite becomes larger and more diverse.

To view an individual test case, one must open up a record and navigate to the correct tab. Thus, there is no good way to 'read' a test suite as a related flow of test cases. The two-level hierarchy becomes restrictive as the number of components and sub-components grows. As new development, touching a wide variety of components is added to the product with subsequent releases, the flow and organization of the test suite can become further compromised. For example, when a patch is issued following a main release, a group of tests will often be written that touch a wide variety of functionality. These tests will usually be grouped together according to the patch rather than placed in the flow of the original set of tests, where they belong contextually. The negative implications of having some tests out of the expected order can be mitigated to some extent by spending time reorganizing the test suites after the completion of a new release, but, as is well-known, time is seldom found for such an activity.

Thus, the use of a test management tool involves a trade-off of test case organization and readability with the management of the test effort. There is no right or wrong answer as to whether the benefits provided by a test management tool will be worthwhile. Compromise solutions exist, such as abstracting the concept of a test case within the test management tool to include a set of related tests, perhaps even all of the tests for a subcomponent of the product. These related tests would then be stored as one unit within the test management tool. In that manner, the number of individual test records can be kept under control, whereas many of the test management benefits of the tool can be maintained. The only definitive directive that can be given to a test organization contemplating the use of a test management tool is to study the tool functionality carefully, and know what they are getting into.

## ***The 'First Principles' Conundrum***

When designing individual test cases, the question arises as to how much prerequisite material to include with each test case. A related question is whether to build each individual test case as a self-contained unit, or to design a set of tests with each subsequent test being dependent on the execution of preceding tests. This will often come down to a question of granularity versus efficiency.

Consider the following simplified set of test case steps:

### **Method 1**

- 1) Add a record
- 2) Modify the record
- 3) Delete the record

## **Method 2**

- 1) Add a record
- 2) Add a record, modify it.
- 3) Add a record, delete it.

Clearly, Method 1 is more efficient as well as easier to read and review than Method 2. However, with Method 2, each test stands on its own as an independent unit. In Method 1, a failure of the Modify may very well affect the success of the third test case. In Method 2, this will not happen, although a failure of the Add will certainly affect the success of all three test cases. Even if test cases are written according to Method 2, an astute tester should be expected to run them as Method 1 in a problem-free case.

Consider the following test case (an extreme example of Method 2):

- Install the program, start the program, run the setup, add the record, delete the record. Check that the record no longer exists.

In most cases, the first three instructions would be considered self-evident. Obviously, common sense must be used. If one is testing the steps of running a subcomponent of a program, it can certainly be assumed that the program is installed and running. If some prerequisites apply to an entire set of test cases, they should be abstracted as such without needless repetition in each test case. Once again, there is no right or wrong answer as to what testing style to use. A balance must be found between providing a full description of the testing activities, including prerequisites, and producing a readable, efficient test case.

Whatever test case style one uses, there are four criteria that must be followed to define a good test case:

- a) A clear, concise title briefly describing the objectives.
- b) A full description of objectives.
- c) A description of prerequisites (could apply to a group of tests).
- d) A concise description of steps and expected results.

## ***Linguistic Style***

In the most abstract definition, a written test case is no different from a piece of literature. Both are vehicles for conveying a set of thoughts from the author to the reader. As with any piece of literature, a wide variety of linguistic variants can come into play. Tests can be written in a formal or informal style. The instructions can be given in the second or third person. Extraneous, motivational material (answering the question 'why') can be included, or the test case can focus on the 'how'. Paragraph or point form can be used. Expected results can be highlighted or buried within the steps.

There may be cases where a more explicit or narrative style format is appropriate; however, in the vast majority of situations, a terse and minimalist style would be preferred. It is often simpler for the tester to use the use case format when writing the test cases, since the 'cut and paste' method can be used to copy the use cases into test cases. More effort is needed to rework the use cases into a style that is appropriate for test cases.

As an illustration of the difference between an explicit style and a terse style, I present the following two versions of a test case.

### **Verbose style:**

The user specifies a name 'John Doe' and an address '123 4th Ave.' to add a new customer record to the file of customers of the XYZ luggage shop. The user clicks the Enter button to store the customer record. The user then closes the program. After the customer calls to cancel his business, the user decides to delete the record. He/she starts the program, enters 'John Doe' to obtain the customer record, and clicks the Delete button. The user then specifies the name 'John Doe'. The user sees a 'Name does not exist' message in the console.

### **Terse style:**

- Add a customer with name 'John Doe' and address '123 4th Ave.' Click Enter to store.
- 2) Close the program.
- 3) Restart the program. Enter 'John Doe' to select the record.
- 4) Press Delete.
- 5) Enter 'John Doe' once again.
- Expected result: 'Name does not exist' message appears in the console.

### ***Instantiation of Data***

The writer of test cases must choose an appropriate depth of description of test data. Consider the following two cases:

### **Full details of data provided:**

- Add a customer with name 'John Doe', address '123 4th Ave.', and telephone number '123-4567'.

### **General description of data:**

- Add a customer. Specify a first name, last name, address, and telephone number.

Once again, there is no right or wrong answer as to which option is preferred. If data validation or the ability to test complex input is being tested, the tester would tend to provide more detail in the data. However, even in that case, full detail need not be provided. It may be best to provide only sufficient detail so as to highlight the point of the test. Excess detail may very well cloud the factor that differentiates the current test case from all others in the suite. Another justification for specific data would be if the data specified for one test case is expected to be used in subsequent tests in the test suite. My preference is to keep the specified data as minimalist as possible in order to ensure that the test objectives are met.

### ***Black Box versus White Box***

Black Box testing refers to a test conducted without the tester using any knowledge of the underlying program structure, whereas White Box testing refers to a test conducted making use of such knowledge. Should a tester use knowledge of program structure to guide in the writing of the test suite, or should the tester rely solely on the user's view of functionality? As expected, both approaches have their place under different circumstances. To some extent, the answer depends on the test phase, with Unit Test tending to white box testing, User Acceptance Test stressing black box testing, and Functional Test somewhere between the two types.

It is in the middle stage test phases, such as Functional Test, where individual decisions will frequently have to be made. Consider a product that features a customer definition page consisting of numerous fields, such as name, address, telephone number, personal notes, etc. This page is invoked from 12 different points in the program. The underlying component is equivalent in each case, with the same program code being executed. In such a case, very little value would be gained by running all of the field validations and error conditions for each of the 12 invocations. A full set of tests should be run for one case, and basic functionality should be tested for all others. (In our example, basic functionality would include a test to see if a customer can be added and deleted from all invocations.) It may be beneficial to run a spot check of other tests for some of the other invocations. Another style would be to distribute the data validation tests among the various invocations. If one does not exercise caution in such cases, the size of a test suite can grow exponentially.

There are other examples where knowledge of component functionality might aid the test case writing process. These can include direct checking of an internally used database for data integrity, intercepting data before it is displayed on a GUI, and testing with debugging code to ensure that proper transaction paths are followed.

## **Summary**

Five different issues of test case style were examined in this paper.

- Storage of Test Cases in a Test Management Tool
- The 'First Principles' Conundrum
- Linguistic Style
- Instantiation of Data
- Black Box versus White Box

The preferred style will depend on various factors, and no definitive answer exists for any of the variants (although the preferences of this author will be obvious in most cases). The writing of test cases is not a mechanical activity, but rather one that involves thought, creativity, decision making, and utilization of effective communication methods. The ultimate success of the tested product will depend to a large degree on the judgment and good sense invested by the author in the test case writing endeavor.

## **Copyright and Trademarks:**

© Copyright: International Business Machines Corporation, 2006. All rights reserved.

IBM, Lotus, Notes, Rational, Rational Software Corporation, and WebSphere are registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

The views expressed herein represent the views of the author, and not necessarily of IBM.