Focus on Quality

Validation of Spectrometry Software

Part VI — Designing Performance Qualification Tests

R.D. McDowall

Many publications and books have been written about computer validation, but several of them provide insufficient information about the design and detail of what tests to carry out in the performance qualification (PQ) phase, how PQ tests are written, and how these tests are documented. This column will discuss how to identify some of the tests to apply to spectrometry software, how to write the test procedures and instructions, and how to execute the actual testing.

R.D. McDowall is principal of McDowall Consulting (Bromley, UK), and "Questions of Quality" column editor for *LCGC Europe*, *Spectroscopy*'s sister magazine. Address correspondence to him at 73 Murray Avenue, Bromley, Kent, BR1 3DJ, UK.

Introduction

In the previous installment in this series, I described the PQ of spectrometry software in general terms (1), including an outline of the PQ test plan sections and the test scripts that specify and document the tests to be executed. I also discussed, in general terms, the three main areas of system functionality that should be tested in a system:

- scientific and instrument functions,
- *21 CFR Part 11* technical controls and functions, and
- backup and preservation of electronic records.

The goal for this installment is to give you more detail about how PQ tests should be designed, written, and executed. Moreover, how should the testing be linked back to the requirements documented earlier in the life cycle for the system that we are validating? Because there are many types of spectrometry software, I obviously cannot go into detail about each one, so I will select one of the functions that should be common to all and outline the principles here. I'll then leave it to you to develop the detailed approach for your specific system.

Testing Software Principles

The principles of software testing are shown in Figure 1. Testing consists of a defining a series of tests from the requirements defined in the approved system requirements specification (SRS). From an individual requirement, a test can be defined for the software; this test will have one or more expected results, along with defined acceptance criteria for the test regarding passing or failing. We will be conducting black box testing: we usually do not know the detailed algorithms employed by the software, only the overall function that the software will perform. At the conclusion of a test, compare the outcome with the acceptance criteria: does it pass or fail?

This is software testing in its simplest form. It is not rocket science, and a chemist with at least a dozen neurons working sequentially can learn this relatively easily when given enough time by management to do so.

But . . . You don't want to test each item individually, do you? Elegance in software testing is how many requirements you can test, either simultaneously or sequentially, in a series of linked test instructions.

The quality of testing is not just deriving tests to pass but also tests to fail. Think about common problems and how the software you are validating will cope with them.

Worst-case testing does not mean the Martians have landed. Be realistic and look at the system you are trying to validate, and design tests based around these types of worst-case scenarios — for example, computation calculations that place a heavy load on the processor, or largest numbers of samples for analysis in a batch. Besides, finding a Martian with knowledge of predicate rules and *21 CFR Part 11* can often take a long time.

Types of Software Testing

Some types of testing that could be carried out include the following:

- Boundary test: the entry of valid data within the known range of a field; for example, a pH value would only have acceptable values within 0–14.

- Stress test: entering data outside of designed limits; for example, a pH value of 15 (This is an example of testing to fail: how will the software cope with this data?).
- Predicted output: knowing the function of the module to be tested, a known input should have a predicted output.
- Consistent operation: important tests of major functions should have repetition built into them to demonstrate that the operation of the system is reproducible.
- Common problems, on both the operational and support aspects of the computer system, should be part of any validation plan. The predictability of the system under these tests should generate confidence in its operation.

Defining, Documenting, and Testing System Security

Requirements traceability is a key issue in current computer validation best practice. If the requirement is not specified, then you have not written the user requirements specification correctly and completely, have you? We'll look at system security, as this should be applicable to all spectrometry software.

Table I. Continuum of user privileges.

Access privilege	Access rights
Zero-level	No access rights, or access denied
Execute only	User can execute functions accessed but nothing else
Read only	User can only read the data accessed; cannot write or append anything
Write only	User can overwrite data
Read–write	User can read or write as required
Append only	User cannot change any data but can add additional information
Administrator	Full access rights to create, read, write, copy, and delete data

Is the Requirement You Are Testing Specified?

The basis for all PQ testing is the system requirements specification and the individual requirements written therein. There is a very simple way to determine if the requirement has been written correctly: can you define a specific test without having to assume anything? If you can, the requirement has been written correctly. If you cannot, the requirement is poorly written and capable of many interpretations. An example of a poorly written function: the system requirement specification states that "the application must have security functions." Explicit tests cannot be derived from that statement.

User Types. More time and effort must be spent defining the various user types that are necessary. Typically this will be a minimum of two, such as a user and a system administrator/supervisor. Your spectrometry software will usually have a security module that the system administrator will configure to allow the different user types access to different functions in the application.

User Privileges. Any discussion of logical security of an application should first consider what each user could do when they use any function. These are the privileges associated with the user of a function within an application. This has a continuum that ranges from the ability to undertake any function, to being denied access. These privileges are shown in Table I, and they are intended to be generic. This continuum may need to be tailored to any spectrometry application in practice; for instance, you may decide that an execute-only function and a read–write are so similar that, in practice, combining them makes sense. Alternatively, the privilege may not be implemented in the application you have purchased or developed.

The functions available to each user type need to be documented as either part of the system requirements specification, or reference needs to be made to a standard operating procedure where this information is located. One way of accomplishing this is to define the requirements in the form of a matrix of function versus user type, as shown in Table II.

Once your system is operational, the system administrator should review access rights of all individual users regu-

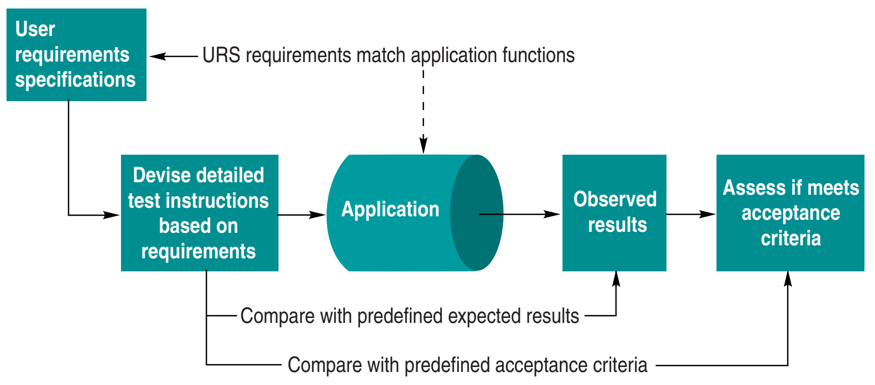
User requirements specifications
URS requirements match application functions
Devise detailed test instructions based on requirements
Application
Observed results
Assess if meets acceptance criteria
Compare with predefined expected results
Compare with predefined acceptance criteria

Figure 1. Principles of designing tests.

larly, especially as they are trained or are promoted — events that will result in a change of user privileges.

Designing the Tests

Now that we have the requirements for access control correctly specified, we can start to design the tests that will demonstrate whether or not the system has been correctly specified and configured.

Look at the definition of access rights for each user type in Table II — you can define a series of tests directly from this table:

- A system administrator can perform all account management and system backup tasks (testing to pass)
- A user cannot perform account management or system backup functions (testing to fail)
- A system administrator can perform all sample analysis tasks (testing to pass)
- A user can perform sample analysis, including modification of methods and running existing report (testing to pass), but cannot create reports or methods (testing to fail).

Companies that come from the Clint Eastwood school of computer validation (graduating as "Do You Feel Lucky?") will assume that the vendor has tested everything, and nothing has to be done.

If the requirements are specified correctly, it is relatively easy to define tests to demonstrate that they function correctly.

Table II. Defining user privileges for each user type.

Software function	User	System administrator
Account management		
Create/modify/disable user accounts		X
Create/modify user types		X
Allocate users to user type		X
Sample analysis		
Create methods		X
Modify methods	X	X
Run samples	X	X
Calculate results	X	
Electronically sign results	X	
Run existing report	X	X
Create report		X
Electronically approve results		X
System backup		
Backup system to tape		X
Recover from tape to system		X

Risk Analysis: Extent of Testing?

Some of you will ask, "How much is enough?" Others will ask the corollary, "What's the minimum I can get away with?" You need to look at the risk involved with the system and the time taken to test. The black (compliant) and white (noncompliant) approaches can be summarized as:

- The simplest way is to test everything: all user types and all functions. No regulatory comeback here, but no work, either.
- Companies that come from the Clint Eastwood school of computer validation (graduating as "Do You Feel Lucky?") will assume that the vendor has tested everything, and nothing has to be done.

Working in the laboratory, you must consider the Good Manufacturing Practice regulations: *21 CFR 211.160(b)*: work has to be "scientifically sound," which comes as a disappointment to many. Therefore, consider the documented arguments that you can produce that can reduce the work of testing the access control functions:

- The vendor has tested the basic software system (better if backed up with a vendor audit report)
- You have configured the software within the boundaries defined and tested by the vendor of the software (configuration has been documented as described in the previous section)
- Therefore, only test your configuration of the system
- Do you test all or just representative functions of the system? Even for high-risk systems, I would suggest that you only test representative functions.

Refining the Test Design

Taking the view that we will only test a representative selection of our configured user functions, we can now develop two test cases:

- A system administrator can create a new user, create a method, analyze a sample, and create a new report
- A user cannot create a new user, method, or report, but they can analyze a sample.

Now that we have designed the test cases, we need to consider how to document them.

PQ Test Documentation

I reiterate that the terminology I use is derived from Institute of Electrical and Electronics Engineers (IEEE) software engineering standards; in your organization, the same things may be called something else.

Key Test Script Sections

You'll remember from the last installment of this series (1) that a test script consisted of a number of sections. The key sections that I want to focus on are:

- Test procedure steps
- Expected results and acceptance criteria
- Actual results

Building from these three areas, in

my view, a test procedure should consist of three main stages:
1. Test steps and expected results (defined before testing starts), observed results, note log, pass–fail statement, and who performed the testing (written contemporaneously as the testing is conducted).
2. Documented evidence. Not specifically required by IEEE standards, but essential for collating information used to support the testing for QA and regulatory inspectors' review.
3. Acceptance criteria. These must be explicitly stated and *not* implied, because they are in many qualification documents.

Documenting Test Execution

Regulatory Viewpoint. The draft *Part 11* validation guidance (2) that was proposed to be withdrawn in February 2003 has a short section on test results:

> **5.4.3 How test results should be expressed.** Quantifiable test results should be recorded in quantified rather than qualified (e.g., pass/fail) terms. Quantified results allow for subsequent review and independent evaluation of the test results.

The question is, how do you interpret this statement? Let's look at my suggested approach and see if you agree.

Documenting Test Execution Instructions and Expected Results. One question often asked is, "How much detail do I need to put into the test execution instructions?" This depends on your company's approach to risk, and the amount of effort they wish to invest. Again, from my perspective, the test steps should be written for a trained user, and not a novice, to execute. This saves writing the embarrassing series of instructions starting with: "Sit in front of the workstation; press the 'on' button and wait for the operating system to boot . . . " Don't laugh; I've seen examples of this.

Therefore, write the instructions so that a trained user can execute them. If a user can come to one, and only one, result, then the test steps have been written correctly; if not, they need more detail. The detail depends on whether this is a new system being validated for the first time, or an existing system being upgraded and (re)validated. Often these two will be different as the user maturity and experience differs; see Tables III and IV to see the differences between these two approaches.

Looking at the test execution instructions in Tables III and IV, one is written in a very terse style for an experienced user and the other in more detail. Which style is appropriate for your laboratory? The detailed style is good if you have high staff turnover and want to retain consistency of execution; the disadvantage is that if the user interface changes, then you could face rewriting the procedures to reflect this.

The terse style has the advantage that it is quick to write, but also easy to assume implicit tasks that are obvious to the writer but not to the person executing the procedure. The issue is, can the implicit steps be remembered the next

Table III. Test script execution instructions written for new users of the application.

Test steps	Expected result
1. The system administrator defines user "Security" as user type "Guest"	User "Security" defined as "Guest"
2. Log onto data system as user "Security"	Access to the system
3. Enter "Configure System" and select "Users"	Function window opens and lists users
4. Select user "Security" and select "Properties"	The properties window for user "Security" opens
5. Check in the General tab the User Type field	User type "Guest" displayed
6. Cancel to leave the Properties window	Leave properties window
7. Attempt to access "Select 'New Method'" from the "Method" menu	Function not available
8. Exit "Configuration Manager"	"Configuration Manager" window is closed
9. Enter "Browse Methods" and select a project Name: ______________	The window for the specified Method opens
10. Verify that the selected Method contains Results	Method contains results
11. Select a result and select "Preview" in the drop-down menu	A window opens, named "Open Report Method"

Table IV. Test script execution instructions written for experienced users of the application.

Test steps	Expected result
1. The system administrator defines user "Security" as user type "Guest"	User "Security" defined as "Guest"
2. User "Security" logs onto the system and looks at his or her access privileges under "Configure System"	User privileges of "Guest" displayed
3. Attempt to access "Select 'New Method'" from the "Method" menu	Function not available
4. Enter "Browse Methods" and select a project Name: ______________	The window for the specified Method opens
5. Access one of the results in the selected Method and report results	A window opens, named "Open Report Method"

Table V. Test script with space to document the observed results and any test execution notes.

Test steps	Expected result	Observed result	Note log	Pass/ fail	Initials
1. The System Administrator defines user "Security" as user type "Guest"	User "Security" defined				
2. User "Security" logs onto the system and looks at his or her access privileges under "Configure Systems"	User privileges of "Guest" displayed				
3. Attempt to access "Select 'New Method' from the "Method" menu	Function not available				
4. Enter "Browse Methods" and select a project Name: _______________	The window for the specified Method opens				
5. Access one of the results in the selected Method and report result	A window opens, named "Open Report Method"				

time the test script must be executed again? This approach is best suited to a multiuser environment where there is sufficient experience to ensure that the scripts are understandable by the user base as a whole, rather than an individual.

Regardless of the approach used, ensure that the test scripts are signed and approved *before* they are executed.

Documenting Testing

Writing Observed Results. When the test steps are executed, we need to know:

- Who executed the test?
- What were the observed results?
- Did the test step pass or fail?

Therefore, these questions need to be incorporated into the outlined test procedures.

In addition to the observed results, there will be supporting evidence of screen shots, printouts, and electronic files.

Unexpected Results. It is rare that a PQ test suite is executed without some issues arising. These issues can come from a number of causes:

- Misunderstood test instructions
- Incorrectly written test instructions
- Expected results written poorly (too detailed or insufficiently detailed)
- Incorrectly set acceptance criteria
- Software error found.

Regardless of the cause, the test script needs to have a way to document and resolve these issues. This is where the note log is useful.

Suggested Documentation. Table V shows an outline that could be used to incorporate the test steps, plus the expected results, plus the four requirements just mentioned. Again, the level of detail that you should go to is a balance between time and effort; however, I would strongly suggest that you write the documentation not only for execution, but also for internal audit and external inspection. The effort will repay itself repeatedly.

Collating Documented Evidence. During the testing of the software, there will be output from the system either in the form of paper printout or electronic records or, for some test steps, both. These need to be retained and collated together as evidence that the test script was executed. To aid audit and inspection, there should be a section in the test script to collate the various documented evidence together.

Has the Test Passed or Failed?

Explicitly Written Acceptance Criteria. A weak point with most test scripts or protocols is the lack of explicitly written acceptance criteria. If acceptance criteria are not written down before execution, how does a tester, the person who will review the results, and even an inspector know if the tests have passed or failed? The unwritten rule is that if the observed results match the expected results, then the test has passed; however, if this is not written down anywhere, this approach is rubbish.

Document and approve the acceptance criteria for each test and compare with the actual ones in a specific section of the test script.

Your test scripts will consist of one or more test procedures; at the end of the document, summarize the results of each of these, then state if the test script passes or fails, and sign this.

Summary

In this column, I have gone through the stages to ensure that the PQ testing is specified and documented correctly. The level of detail and risk is left to the laboratory to determine, but many labs can't or won't find the time and effort to do the job correctly the first time. After a visit from the regulators, the time and effort is freely available the second time around. Do the job properly, because compliance is always cheaper the first time.

In the next installment, we will look at writing the validation summary report to document the whole of the validation effort before the system is released for operational use.

References

1. R.D. McDowall, *Spectroscopy* **18**(4), 26–30 (2003).
2. "FDA Draft Guidance for Industry, *21 CFR 11* Electronic Records and Electronic Signatures Validation," Food and Drug Administration, Washington, DC, 2001. ■