Functional Testing

Rex Black, President, RBCS, Inc.

Functional testing focuses on what the system does, rather than how it does it. Non-functional testing is focused on how the system does what it does. Both functional and non-functional testing are black-box tests, being focused on behavior. White-box tests are focused on how the system works internally—i.e., on its structure.

Functional tests can have, as their test basis, the functional requirements. These include both the requirements that are written down in a specification document and those that are implicit. The domain expertise of the tester can also be part of the test basis.

Functional tests will vary by test level or phase. A functional integration test will focus on the functionality of a collection of interfacing modules, usually in terms of the partial or complete user workflows, use cases, operations, or features these modules provide. A functional system test will focus on the functionality of the application as a whole, complete user workflows, use cases, operations, and features. A functional system integration test will focus on end-to-end functionality that spans the entire set of integrated systems.

The test analyst can employ various test techniques during functional testing at any level. All of the techniques discussed in Advanced Software Testing: Volume 1 will be useful.

We should keep in mind that test analyst is a role, not a title, job description, or position. In other words, some people play the role of test analyst exclusively, but others play that role as part of another job. So, when dedicated, professional testers do functional testing, they are test analysts both in position and in role. However, when domain experts do the analysis, design, implementation, or execution of functional tests, they are working as test analysts. When developers do the analysis, design, implementation, or execution of functional tests, they are working as test analysts.

For test analysts in the ISTQB Advanced syllabus, we consider functional and usability testing as concerned with the following quality attributes:

- Accuracy
- Suitability
- Interoperability

Copyright © 2015, RBCS, All Rights Reserved

- Usability
- Accessibility

In this excerpt, we'll look at testing the first three of these attributes, starting with accuracy.

Functional accuracy testing is concerned with adherence to specified or implied functional requirements. In other words, does the system give the right answer and produce the right effects? Accuracy, in this case, also refers to the right degree of precision in the results.

Functional accuracy testing can include tests of computational accuracy. Indeed, for any application that is used for math, statistics, accounting, science, engineering, or other similar math-intensive functionality, testing of computational accuracy is critical. We want to make sure that all data and situations are handled correctly. Accuracy testing is important at all stages in the lifecycle and in all test levels. To test accuracy properly, you must have a reliable and precise test oracle, which can include specifications, legacy systems, competitors' systems, and more.

Let's look at an example, using Quicken's stock buy/add screen, specifically the number of shares, the price per share, and commission fields. We can apply equivalence partitioning and boundary value analysis to these fields, which identifies thirteen specific input values for each field.

In this case, as shown in Figure 1, we would also want to add testing of the total cost field. This is a calculated output field. It is calculated by using the three input fields. As we can see in Figure 1, there is something not right with the calculation. The combination of the maximum number of shares and the maximum price per share is not giving us the right result in the total cost field. Or perhaps the number is right, internally, but is overflowing the display space. Either way, I'd report this as a bug.

 Copyright © 2015, RBCS, All Rights Reserved

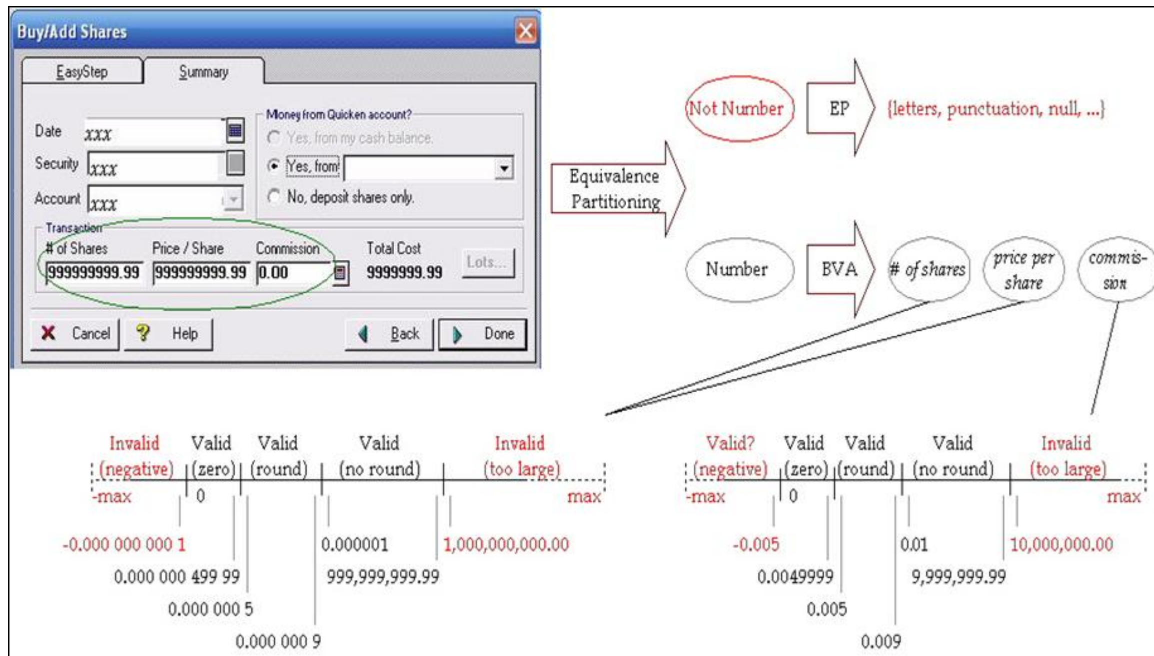

Figure 1 Accuracy testing example

Moving on to functional suitability testing, this is focused on the appropriateness of a set of functions, relative to its intended, specific tasks. In other words, given the problem we need to solve, can the system solve it?

Notice that there is an element of validation to this focus. We are intent on demonstrating the value of the system in some specific situation. A shorthand way of thinking about validation is: "Are we building the right system?"

This is in contrast to verification, which is about following the right process, having traceability between functions and requirements and between tests and requirements, and using that traceability to show that the requirements are met. The shorthand way of thinking about verification is: "Are we building the system right?"

Suitability testing starts, at the earliest, during integration testing, as you need to have enough of the system present for the system to actually solve a real-world problem. Suitability testing usually continues through system test and into acceptance test. For example, in Agile projects, the feature demo with the product owner, the tester, and the developer is a form of suitability testing. If the user's needs are not met, regardless of whether requirements are fulfilled, then the software is not suitable.

> Based on what we're trying to accomplish here, it's clear that we need to test in ways that strongly resemble actual workflows. We can employ use cases, test scenarios, and exploratory testing. Other techniques tend to be a bit too

 Copyright © 2015, RBCS, All Rights Reserved

fine-grained or distracted by bug hunting to serve the purpose. Indeed, you have to be careful with exploratory testing to make sure that the charters reflect the need to explore real-world usage of the product.

Let's look at an example, using a an informal e-commerce purchase test derived from a use case shown in Figure 2. This use case says that we should be able to put items in a shopping cart, initiate a checkout, enter the information the system needs to process the purchase, and confirm the purchase before it's done.

The use case also says that the system needs to reject attempts to check out with an empty cart, to reject invalid inputs in the purchase information, and to recognize an abandoned cart when it sees one.

In Table 1, we test the suitability of the system's functionality to handle the typical workflow. We had an implied requirement to accept all four major credit cards, so we tested those. We also had an implied requirement to support both US and international customers, so we tested those.

E-commerce purchase: Normal workflow

1. Customer places one or more Items in shopping cart
2. Customer selects checkout
3. System gathers address, payment, and shipping information from Customer
4. System displays all information for User confirmation
5. User confirms order to System for delivery

Exceptions

- Customer attempts to checkout with empty shopping cart; System gives error message
- Customer provides invalid address, payment, or shipping information; System gives error messages as appropriate
- Customer abandons transaction before or during checkout; System logs Customer out after 10 minutes of inactivity

Figure 2 Suitability testing example: use case

#	Test Step	Expected Result

 Copyright © 2015, RBCS, All Rights Reserved

1	Place one item in cart.	Item in cart
2	Click “Check out.”	Checkout screen
3	Input valid US address, valid payment using American Express, and valid shipping method information.	Screens display correctly; valid inputs accepted
4	Verify order information.	Shown as entered
5	Confirm order.	Order in system
6	Repeat steps 1–5, but place two items in cart, pay with Visa, and ship internationally.	As shown in steps 1–5
7	Repeat steps 1–5, but place the maximum number of items in cart and pay with MasterCard.	As shown in steps 1–5
8	Repeat steps 1–5, but pay with Discover.	As shown in steps 1–5

Table 1 Suitability tests (typical)

In Table 2, we test the suitability of the system’s functionality under exceptional conditions. We check the empty cart. We check the inputs of invalid information on all screens, including the ability of the system to stop us from proceeding unless a form is correctly filled. Finally, we test the abandonment of a cart from all possible screens. We have clear traceability from the tests back to the use case steps.

#	Test Step	Expected Result
1	Do not place any items in cart.	Cart empty
2	Click “Check out.”	Error message
3	Place item in cart; click “Check out;” enter invalid address, then invalid payment, then invalid shipping information.	Error messages; can’t proceed to next screen until resolved
4	Verify order information.	Shown as entered
5	Confirm order.	Order in system
6	Repeat steps 1–3, but stop activity and abandon transaction after placing item in cart.	User logged out exactly ten minutes after last activity

 Copyright © 2015, RBCS, All Rights Reserved

7	Repeats steps 1–3, but stop activity and abandon transaction on each screen.	As shown in 6
8	Repeat steps 1–4; do not confirm order.	As shown in 6

Table 2 Suitability tests (exception)

Finally, let's look a functional interoperability, which involves testing the ability of systems or components to exchange information and use that information, in all intended environments: Environments refer to hardware, of course, but also software, middleware, connectivity infrastructure, database systems, and operating systems. This would include not only elements of the environment that the system must interoperate directly with, but also those with which it interoperates indirectly.

As you might imagine, there's a major test configuration element involved with understanding the test environments needed. The environments are then tested with selected major functions. If you suspect that particular functions might interact in particular ways with particular test environments, be sure to test those. If not, then we can test arbitrary combinations of functions with environments.

Interoperability is, of course, about systems interacting with each other. Good interoperability implies ease of integration with other systems with few if any major changes.

Design features can raise important considerations for testing software interoperability. Examples include the following:

- The system use of industrywide data or communications standards, such as XML
- The ability of the system to provide standard, flexible, and robust interfaces
- The ability of the system to automatically detect and adapt to various interfaces, communication speeds, protocols, and the like

Since these are design issues, you may need to consult design specifications as well as requirements specifications.

As a test analyst, you can expect to do a lot of interoperability testing when you are developing or integrating commercial off-the-shelf (COTS) software and tools. That's also true if you are developing systems of systems from off-the-shelf or custom-developed applications. Interoperability is important during component integration testing, system testing, and system integration testing.

For testing of functional interoperability, especially end-to-end functionality, you can employ use cases and test scenarios. To determine the environments, you can

 Copyright © 2015, RBCS, All Rights Reserved

use equivalence partitioning when you can understand the possible interactions between one or more environments and one or more functions. When interactions are not clear, you can use pairwise testing and classification trees to generate somewhat more arbitrary configurations. Decision tables can be useful to identify conditions that interact, and state transition diagrams may apply with stateful interfaces.

Let's look at an example, by combining the use case example that we just saw with pairwise testing. You can use an orthogonal array or a pairwise testing tool to create a pairwise table of tests.[1] In terms of environment, we have four factors:

- Connection, either WiFi (slower) or wired Ethernet (faster)
- Operating system, either Mac, Linux, Windows 7, or Windows 10
- Security software, either operating system default settings, Symantec, Trend Micro, or PCMatic
- Browser, either Firefox, Internet Explorer, or Chrome

Looking at the use case, we see four typical usages:

- The first is American Express, to purchase one item, for shipping in the US. Let's call that usage A.
- The second is Visa, to purchase two items, for shipping internationally. Let's call that usage B.
- The third is MasterCard, to purchase as many items as the cart will hold, for shipping in the US. Let's call that usage C.
- The fourth is Discover, to purchase one item, for shipping in the US. Let's call that usage D.

In Table 3, you can see the combination of tests with the environments. To execute the test, you first obtain the correct configuration, and then you run the usage derived earlier from the use case.

Test	Connection	OS	Security	Browser	Usage
1	WiFi	Mac	OS	Firefox	A
2	WiFi	Linux	Symantec	IE	B
3	WiFi	W7	Trend	Chrome	C
4	WiFi	W10	PCMatic	~	D
5	Wired	Mac	Symantec	Chrome	D

[1] I suggest the free tool available from the National Institute of Standards and Technology, ACTS, which you can find at www.nist.gov.

 Copyright © 2015, RBCS, All Rights Reserved

6	Wired	Linux	OS	~	C
7	Wired	W7	PCMatic	Firefox	B
8	Wired	W10	Trend	IE	A
9	~	Mac	Trend	~	B
10	~	Linux	PCMatic	Chrome	A
11	~	W7	OS	IE	D
12	~	W10	Symantec	Firefox	C
13	~	Mac	PCMatic	IE	C
14	~	Linux	Trend	Firefox	D
15	~	W7	Symantec	~	A
16	~	W10	OS	Chrome	B

Table 3 Pairwise testing of interoperability

The technique would be similar for the negative tests. Remember that we don't want to mix negative and positive tests, especially here, because we're testing for the ability of the e-commerce system to complete an entire function—a purchase—on various supported environment configurations. Notice also that these sixteen tests cover the suitable tests shown earlier, so you would not need to run those tests separately.

 Copyright © 2015, RBCS, All Rights Reserved