28

December 2014

# testing experience

The Magazine for Professional Testers

**The Three Pillars of Agile Quality and Testing**
*by Robert Galen*

**Testing the Internet of Things – The Future is Here**
*by Venkat Ramesh Atigadda*

***... and many more***

printed in Germany

print version 8,00 €

free digital version

www.testingexperience.com

ISSN 1866-5705

*By Rik Marselis & Bert Linker*

# Organize Your Testing Using Test Varieties and Coverage Types

## The Struggle

Many test managers often struggle to define the proper way to spread the testing efforts throughout the project or release activities in such way that it properly reflects the constraints of quality, risk, time, and cost.

In recent years, the rise in approaches that use short cycles has made it even harder to create a balanced approach to testing and translate that into a test strategy – if there is one at all.

We wondered what the reasons for this problem are. In our opinion, one of the causes lies with confusion about, or even ignorance of, the meaning of the terms "test level", "test type", and "test design technique".

In this age of agile methodologies, "test levels" are often associated with hierarchies in testing, and since Agile promotes doing all activities by one team in a single iteration, there is no hierarchy. The same reasoning goes for "test types". Because all testing happens within the iteration (which Scrum calls a sprint), the people involved want to rush their work and do not want to be bothered with the differences between various types of testing.

Does it really matter? Well, yes! In a survey amongst about 300 projects over the last 5 years, almost 50 % of the people involved said that the quality delivered by agile IT teams was no better than before they adopted Agile. So we need to give extra attention to quality, since quality is supposed to be key in Agile.

## Test Varieties

When it comes to testing, as one of the quality measures that can be taken, we want to make things easy to grasp by introducing something new: the "test variety". This simple term intends to emphasize to all people involved that testing is not a one-size-fits-all activity. Even when all testing activities are done by one team within a single iteration, you will still need to vary the testing. The first variety of testing, of course, is static testing, i.e., reviewing documents and source code. Static testing can both be manual (using techniques like technical review or inspection) and automated (with tools such as static analyzers).

The next view on test varieties relates to the parties involved. The developers have a technical view of testing, looking to see whether the software works as designed and properly integrates with other pieces of software. The designers want to know whether the system as a whole works according to their specifications (whatever the form and shape of these specifications may be). And the business people simply want to know whether their business process is properly supported. Now, during these various tests there will be different points of view, for example related to quality attributes. Functionality will seldom be forgotten in testing, but what about looking at maintainability by the developers, installability by the designers, and usability by the business people?

So in this simple example we can already see at least seven test varieties. During the start-up of a project (for example in a "sprint zero"), a test strategy for the project is established. And for each iteration the test strategy is tuned to the needs in that cycle. This way, all team members know what their points of focus must be during this iteration. By the way, please be aware that when we say "test strategy" we do not necessarily refer to a document, we merely want to emphasize that there must be an agreed way of assigning the right testing activities with the right test intensity. And by being aware of the necessary test varieties you will also have less difficulty in deciding what testing activities can be done within the sprint and what has to be organized separately (the common agreement nowadays is that an end-to-end-test cannot be done by one agile team within their sprint, so this is a test variety that will often be organized separately).

This is the first step to better testing. By making the people involved aware of the relevant varieties of testing, it defies the one-size-fits-all mentality often seen in testing.

## Experience-Based and/or Coverage-Based Approach

The next step in completing the test strategy is to define the proper approach for the test varieties. Based on the desired quality level and the perceived risk level, and within the limitations of time and cost, the team members choose an experience-based and/or coverage-based approach to testing.

Here is another area of testing that many people struggle with. There are very many so-called test design techniques. But, in practice, most testers do not formally apply any technique at all, they just "guess" the best test cases in their specific situation. One reason for this is that there are so many possibilities that they decide not to choose at all. In our opinion, the choice does not need to be hard. In any given situation you only need a simple choice of approaches and about a handful of coverage types to be able to do proper testing.

We distinguish two approaches: experience-based and coverage-based.

For experience-based testing there are a choice of possibilities, of which exploratory testing is the most well-known and appropriate approach. These tests are effective at finding defects, but less appropriate for achieving specific test coverage, unless they are combined with coverage types.

| Coverage Type Group | Coverage Type | Description | Variation |
|---|---|---|---|
| Process | Algorithm | Testing the program structure. | • Statement coverage<br>• Decision coverage (branch testing/arc testing) |
| | Paths | Coverage of the variations in the process in terms of combinations of paths. | • Test depth level 1<br>• Test depth level 2<br>• Test depth level N |
| | Right paths/fault paths | Checking both the valid and invalid situations in every defined error situation. An invalid situation (faulty control steps in the process or algorithm that precede the processing) should lead to correct error handling, while a valid situation should be accepted by the system without error handling. | • Right paths only<br>• Right paths and fault paths |
| | State transitions | Verification of relationships between events, actions, activities, states, and state transitions. | • 0-switch<br>• 1-switch<br>• N-switch |
| Conditions/decisions | Decision points | Coverage of the various possibilities within a decision point with the purpose of arriving at the outcomes of TRUE or FALSE | • Condition coverage<br>• Decision coverage<br>• Condition/decision coverage<br>• Modified condition/decision coverage<br>• Multiple condition coverage (per decision point or across decision points)<br>• Cause-effect graph<br>• Pairwise testing |
| Data | Boundary values | A boundary value determines the transfer from one equivalence class to another. Boundary value analysis tests the boundary value itself plus the value directly above and directly below it. | • Light (boundary value + one value)<br>• Normal (boundary value + two values) |
| | Equivalence classes | The value range of a parameter is divided into classes in which different system behaviour takes place. The system is tested with at least one value from each class. | • One value per class<br>• Combination with boundary values |
| | CRUD | Coverage of all the basic operations (create, read, update, delete) on all the entities. | |
| | Data combinations | Testing of combinations of parameter values. The basis is equivalence classes. A commonly used technique for data combinations is the classification tree. | • Right paths/fault paths<br>• One or some data pairs<br>• N-wise (e.g. pairwise)<br>• All possible combinations |
| | Data flows | Verifying information of a data flow that runs from actor to actor, from input to output. | |
| | Right paths/fault paths | Checking both the valid and invalid situations in every defined error situation. An invalid situation (certain values or combinations of values defined that are not permitted for the relevant functionality) should lead to correct error handling, while a valid situation should be accepted by the system without error handling. | |
| Appearance | Heuristics | Evaluation of (a number of) usability principles. | |
| | Load profiles | Simulation of a realistic loading of the system in terms of volume of users and/or transactions. | |
| | Operational profiles | Simulation of the realistic use of the system by carrying out a statistically responsible sequence of transactions. | |
| | Presentation | Testing the layout of input (screens) and output (lists, reports). | |
| | Usability | Validating whether the system is easy to use, understand, and learn. | • Alpha testing<br>• Beta testing<br>• Usability lab |

*Table 1. Overview of the coverage type groups, examples of coverage types, and possible variations*

Coverage-based testing uses coverage types. A coverage type focuses on achieving a specific coverage of quality and risks, and on detecting specific types of defects. Thus a coverage type aims to cover certain areas or aspects of the test object. Our starting point is that coverage types not only indicate what is covered, but also provide directions on how to do so. Coverage types are, as such, the foundation of the many test design techniques.

## Experience-Based Approach

Below we describe three examples of experience-based testing that may be considered.

### Error Guessing

The tester uses experience to guess the potential errors that might have been made and determines the methods to uncover the resulting defects. Error guessing is also useful during risk analysis to identify potential failure modes. Part of this is "defect-based testing", where the type of defect sought is used as the basis for the test design, with tests derived systematically from what is known about the defect. Error guessing is often no more than ad hoc testing, and the results of testing are totally dependent on the experience and skills of the tester.

### Checklist-based

The experienced tester uses a high-level list of items to be noted, checked, or remembered, or a set of rules or criteria against which a product has to be verified. These checklists are built based on a set of standards, on experience, and on other considerations. A checklist of user interface standards used as the basis for testing an application is an example of checklist-based testing.

Checking of individual elements is often done using an unstructured list. Each element in the list is directly tested by at least one test case. Although checklist-based testing is more organized than error guessing, it is still highly dependent on the skills of the tester, and the test is only as good as the checklist that is used.

### Exploratory

Exploratory testing is simultaneous learning, test design, and test execution. In other words, exploratory testing is any testing to the extent that the tester actively controls the design of the tests, as those tests are performed and use information gained while testing to design new and better tests. Good exploratory testing is timeboxed – based on a charter that also defines scope and special areas of attention. Since exploratory testing is preferably done by two people working together and who apply relevant coverage types for the specific situation at hand, this approach is preferred over the alternatives mentioned above.

### Hybrid approaches

In practice, the use of hybrid approaches is very common. Exploratory testing, for instance, can be very well combined with the use of coverage types. And there are test design techniques that may be used within experience-based as well as coverage-based testing, such as the data combination test (which uses classification trees).

## Coverage-Based Testing

In our experience many testers have difficulty in selecting the proper coverage in a specific situation, which is often caused by confusion about the coverage type that best matches the specific situation they want to test. That's why for coverage based testing we have created four groups of coverage types. Analyse the type of situation you're in and select a coverage type from the group that matches this challenge.

### Process

Processes can be identified at several levels. There are algorithms of control flows, event-based transitions between states, and business processes. Coverage types like paths, statement coverage, and state transition coverage can be used to test (variations in) these processes.

### Conditions/Decisions

In every IT system there are decision points consisting of conditions, where the system behavior differs depending on the outcome of such a decision point. Variations of these conditions and their outcomes can be tested using coverage types like decision coverage, modified condition/decision coverage, and multiple condition coverage.

### Data

Data starts its lifecycle when it is created and ends when it is removed. In between, the data is used by updating or consulting it. This lifecycle of data can be tested, as can combinations of input data, and the attributes of input or output data. Some coverage types in this respect are boundary values, CRUD, data flows, and data combinations.

### Appearance

How a system operates, how it performs, and what its appearance should be are often described in non-functional requirements. Within this group we find coverage types like heuristics, operational and load profiles, and presentation.

## Coverage Type Table

The Table 1 gives an overview of the coverage type groups, examples of coverage types, and possible variations.

Although the overview is extensive, it is not exhaustive. Looking at what can be covered, we could have added aspects like syntax (using a checklist), semantics and integrity rules (using decision points), authorisation, privacy etc. (using checklists, doing reviews, etc.). However, we do not want to over-complicate things. We advise you to check relevant literature for the coverage types and test design techniques that are suitable in your specific situation.

## Test Intensity Table

A main goal of the test strategy is to define the necessary intensity of the testing, commonly based on risk. High risk requires thorough testing, low risk may need only light testing. To give you a practical overview of the coverage types you can select for the different classes, we have highlighted the most commonly used coverage types and some test design techniques in which they can be applied. We have not given an overview for appearance, since the coverage types for appearance are highly interlinked with the aspect to be tested, and we believe that giving a simplified overview would be misleading.

| Coverage Type Group | Test Intensity | | |
|---|---|---|---|
| | Light | Average | Thorough |
| Process | Statement coverage and paths test depth level 1 – process cycle test | Decision coverage and paths test depth level 2 – process cycle test | Paths test depth level 2 – algorithms Test and paths test depth level 3 – process cycle test |
| Conditions | Condition decision coverage – elementary comparison test | Modified condition decision coverage – elementary comparison test or condition decision coverage – decision table test | Multiple condition coverage elementary comparison test or multiple condition decision coverage – decision table test |
| Data | One or some data pairs – data combination test | Pairwise – data combination test | N-wise or all combinations – data combination test |

*Table 2. Test intensity table*

## Conclusion

Applying an effective and efficient way of testing does not need to be bothersome. Using test varieties, a combination of experience-based and coverage-based testing, and your choice of about five coverage types that are relevant for your situation, testing in these fast-paced times will focus on establishing the stakeholders confidence without tedious and unnecessary work. ■

## Literature


- *Testing Embedded Software*, Bart Broekman & Edwin Notenboom, Addison Wesley, 2003, ISBN 9780321159861
- *TMap NEXT for result-driven testing*, Tim Koomen, Leo van der Aalst, Bart Broekman, Michiel Vroon, UTN Publishers, 2006, ISBN 9072194799
- *TMap NEXT in Scrum*, Leo van der Aalst & Cecile Davis, Sogeti, 2012, ISBN 9789075414646
- *Neil's Quest for Quality; A TMap HD Story*, Aldert Boersma & Erik Vooijs, Sogeti, 2014, ISBN 9789075414837


## > about the authors

**Rik Marselis** is one of Sogeti's most experienced management consultants in the field of quality and testing. He is a well-known author, presenter, and trainer who has assisted many organizations throughout the world in actually improving their testing and thus achieving fit-for-purpose quality and increased business success.
Twitter: *@rikmarselis*

**Bert Linker** is an experienced test consultant within Sogeti. He is (co)author of several books and trainer on many test subjects. He has helped many organizations in traditional and agile environments improve their testing and quality processes.

Both Bert and Rik wrote several building blocks for the new TMap HD book that was presented on 28 October 2014.