te
testing
experience

The Magazine for Professional Testers

Advanced Testing Techniques

ignite GERMANY 2010 Conferences Special

Test 2.0 - Advanced Testing Techniques in Life Insurance

A New Quality in Mathematical Testing of Portfolio Management Systems in Life Insurance

by Jens Fricke & Thomas Niess

The demands on mathematical testing of portfolio management systems are increasing: more than ever before testing requires a structured and systematical approach. The reasons for this are ever-shortening product life cycles, annually changing surplus parameters, new legal regulations and increasingly complex products.

The testing procedures used in insurance up to now do work. They can, however, be optimized significantly by structured test processes according to the ISTQB testing process and standardized documentation.

Possible improvements are described below according to the V-model and the ISTQB testing process.

Mathematical Testing

The software component mathematics (mathematical calculation core, actuarial sub-system, calculation model, and so on) is the subject of mathematical testing of portfolio management systems in the insurance industry. It comprises all mathematical values of the portfolio management system.

These values emerge when processing a business transaction (information, technical change, update), in printed material (policies, updates of changes and dynamics), in interfaces (collection/payment interface, printing interface and so on) and in persistent values in the databases (see Figure 1). Thus mathematical testing delimits itself from technical testing (test of total functionality) and the testing of peripheral systems (collection/payment, commission).

Mathematical testing of portfolio management systems basically checks the quality characteristic functionality. It comprises all characteristics that describe the required capabilities of the system. The objective of the test is to prove that each required capability was realized in the system in such a way that the input/output behavior or a specified reaction is fulfilled. According to ISO standard 9126 the characteristic functionality consists of the sub-characteristics suitability, accuracy, interoperability, compliance and security.[1] Interoperability and security are not relevant for the testing of the software component Mathematics. They are the objectives of the technical testing. For mathematical testing the sub-characteristics suitability (each required capability exists in the system and was realized in a fitting way), accuracy (the system provides the correct or specified reactions or effects) and compliance (the software meets the generally acknowledged actuarial principles) are important.

Tests have to be executed in a structed way and systematically, because testing a software system can never be complete due to the multitude of possible combinations. This is the only way to apply limited resources to specific targets, to detect as many failures as possible with reasonable effort and, if possible, to avoid unnecessary testing (testing that does not lead to new findings). The procedures to derive test cases can ideally be combined with the experience of the testers and typical cases that occur in actual production.

1. Test Levels in Mathematical Testing during the Software Life Cycle

The V-model is very well suited to contemplate the test levels in the context of the development/maintenance of a portfolio management system. The levels relevant for mathematical testing are the steps com-
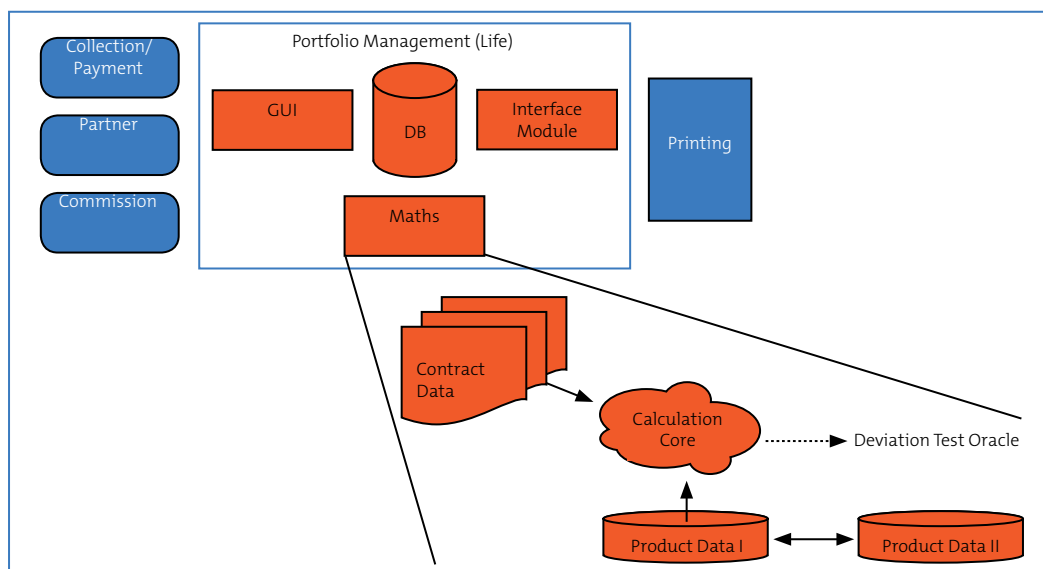

Figure 1: Mathematical Testing in the Context of a Portfolio Management System
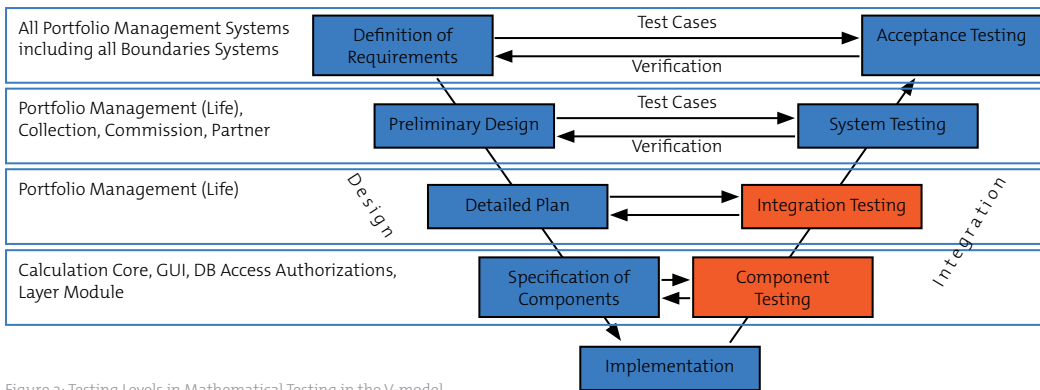
1   See ISO/IEC 9126

Figure 2: Testing Levels in Mathematical Testing in the V-model

ponent testing and integration testing. The other test levels - system testing and acceptance testing - play only a minor role. REF _Ref246816059 \h Figure 2 shows the steps of the V-model and their significance in the life insurance industry. The levels relevant for the mathematical test are indicated in orange.

1.a. Component Testing

The test level component testing subjects the partial system mathematics (calculation core, actuarial subsystem, and the like) as a whole to systematical testing for the first time. Characteristically, the separate software module mathematics is tested without the complete administration system. Thus influences from outside of the component are excluded. A detected failure can be attributed directly to the component tested.

Component testing is the first level in testing after implementation, therefore testing is closely connected to development in this level. Processing test cases without the complete administration system requires a test driver. The development of this driver requires the knowledge of developers. This is why component testing is often called developer testing, even though the tests are not executed by the developers themselves.

The most important objective of component testing is that the test object realises the functionality demanded in its specification correctly and completely (functional test). To check the correctness and completeness of the implementation, the component is subjected to a number of test cases. Typical software defects of functional component testing are calculation errors, missing program paths or the selection of wrong paths. Testing for robustness is another important aspect of component testing although it does not concern mathematical values. This test is executed in analogy to functional component testing. Test input consists of values that are invalid according to the specifications, and suitable exception handling is expected from the component. These negative tests are to prevent the complete system from crashing when an error occurs in one component.

In modern portfolio management systems products and tariffs usually are defined using numerous parameters (cost and return sets, formulas) called product data. The systematical test of these settings can also be assigned to component testing. Here, the component product data (product data definition system or the like) is subjected to static testing, i.e., data is tested without running the software.

A manual check of the product data requires a lot of time and effort. The results of the analysis usually have a lesser quality than results that were obtained automatically. This is caused both by the amount of the product data (products, tariffs, modules, surplus parameters) and by their complexity. Support by a suitable tool, called product data reconciliation, can be helpful. Typical tasks in product data reconciliation are finding out whether there are two similar products or tariffs (e.g., old and new tariff set), or whether two different product data versions (earlier development and current release) vary in specified changes only. In this tool-supported analysis two products or tariffs or two different versions of product data (e.g., before and after realisation of cer-

tain requirements) are compared automatically and the results are output in a suitable way. Next, it is checked whether these differences can be explained by the implemented requirements. The tasks mentioned can be automated using database-based tools. The testing instrument product data reconciliation can complement or even partially replace tests for new profit declarations or product generation.

Component testing can be executed in the form of white-box testing, because it is closely connected to development and the tester often has access to the source code. However, consisting of various complex components the mathematical component itself is too large to design test cases using white-box procedures (statement, branch or path coverage) with justifiable effort. This will only be possible for selected components. In practice mathematical component testing is almost exclusively executed in the form of black-box testing. The test coverage of the test cases specified this way can be measured using tools and complemented by further test cases for those parts of the code that have not been executed so far.

1.b. Integration Testing

In integration testing the modules verified in component testing are tested with regard to their interaction. Integration testing assumes that the test objects transferred (the separate components) were already tested and faults corrected. It is tested whether the interaction of the mathematical component with the remaining administration system works correctly. The objective of integration testing is to find defects in the interfaces and in the interaction of the mathematical component with the rest of the administration system.

In this test level tests can be executed on the complete administration system without peripheral systems for the first time. Therefore this test level also requires test drivers or dummies to provide test data from the peripheral systems (i.e. partners) for the administration system. Here, it has to be ensured that values verified in component testing are correctly processed in the complete portfolio management system, e.g. the way they are displayed on the interface GUI (see Figure 3).
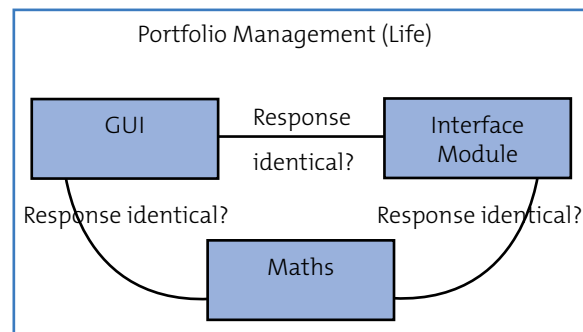

Figure 3: Tests in Integration Testing

Problems can already occur during integration. Problems that are more difficult to find concern the interaction of the components and can only be discovered in dynamic testing. They include, for example, missing or syntactically wrong data which result in a component not working or crashing. Wrong interpretation of transferred data or data transfer at an incorrect or late point in time are other possible problems. These faults manifest themselves in the interaction of the components and thus cannot be detected during component testing.

In practice it often happens that mathematical component testing is dispensed with, and instead all test cases are executed in

integration testing. The disadvantages in this are:

- Most of the failures detected are caused by functional defects in separate components. Implicit component testing is executed in an unsuitable test environment.

- Some failures cannot be provoked, because there is no access to the separate components, thus many defects cannot be detected.

- When a failure or malfunction occurs during testing, it can be difficult to isolate the point of origin and thus the cause.

The reverse case, foregoing integration testing and processing all test cases in mathematical module testing, also exists in practice. The disadvantages of this procedure have already been discussed above.

1.c. System Testing

System testing is of minor importance for mathematical testing. In it, it has to be examined whether the values of the peripheral systems (like printing, provision or monetary transactions), that were verified in the previous test levels, are processed correctly. The testing of mathematical values in this test level is limited to comparing whether the correct data were accessed, because the values used here were already verified in component testing or integration testing (see Figure 4).
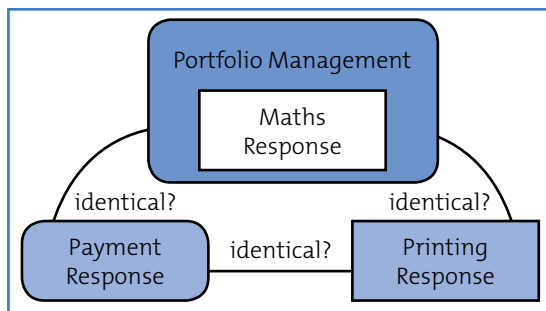
Figure 4: Test in System Testing

An exception is testing in billing and financial accounting. This testing can only be executed in test level system testing because it requires programs for further processing (outside of the portfolio management system).

1.d. Acceptance Testing

In acceptance testing the complete (possibly cross-divisional) IT environment is tested for acceptance. In this test level mathematical testing is not relevant anymore. The tests cases relevant for acceptance testing were already executed in the previous test levels.

1.e. Regression Testing

Regression testing is to ensure that modifications in the portfolio management (enhancement, correction of defects) do not have any undesired effects on the functionality. Here test cases are processed in different versions of the portfolio management system, and the output of the systems is compared. The evaluation whether a test case was successful is not based on a comparison with the specifications but on the comparison of the newly created output to that of the previous version or the output that up to now was regarded as correct. If there are no differences, the regression test case was successful. In case of differences, these have to be analysed. If the differences are wanted, the changed behaviour has to be defined as target behaviour (reference case) for future regression testing. If there are differences that are not desired, the defect has to be localised and corrected.

Regression testing can be executed as a diversifying test[2], i.e.

against a previous version, or as a function-oriented test (against references, i.e. the specifications). Testing against references is to be preferred to diversifying testing, because it can reduce the number of defects creeping in. For the analysis of deviations, diversifying regression testing can be a helpful addition. It can clarify whether a deviation occurred for the first time in the current release or just was not detected up to now.

Regression testing lends itself in particular for mathematical component testing. Furthermore, regression testing of the complete portfolio management system and a comparison of mathematical values make sense. This requires a close coordination between mathematical and technical testing.

Regression testing is an indispensable part of the development and maintenance of a portfolio management system, because portfolio management systems are permanently being enhanced (new functionalities, new product, legal requirements).

2. The Fundamental Test Process

Figure 5 shows the testing process[3] developed by the ISTQB (International Software Testing Qualification Board) and relates the separate activities to the test documents according to standard IEEE 829[4].
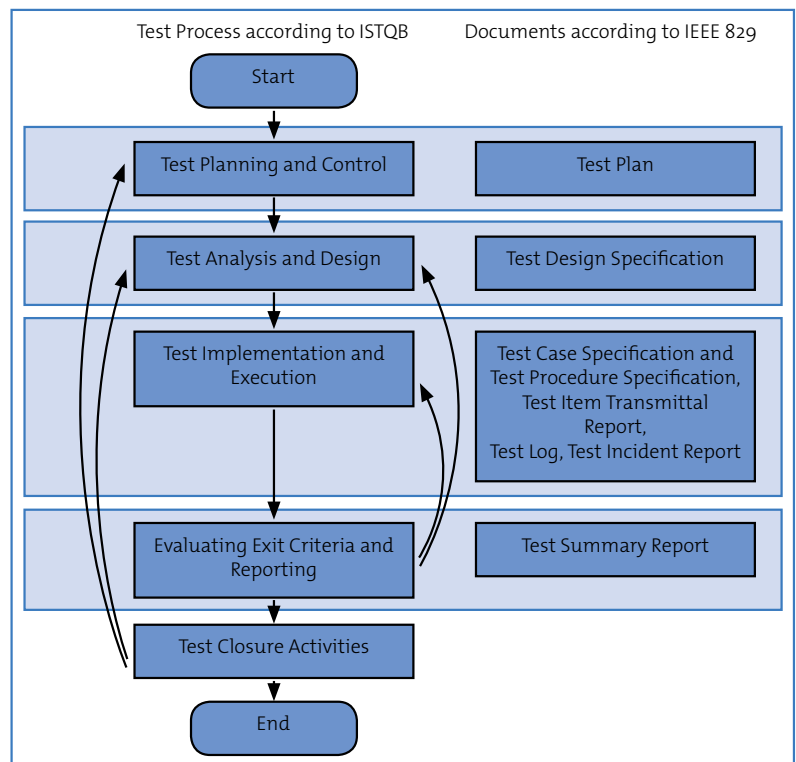
Figure 5: Test Process according to ISTQB and Corresponding Documents according to IEEE 829

2.a. Test Planning and Control

In test planning and control the project manager has to clarify and record what has to be tested, what the objectives of the tests are, when testing is to be started, where testing is to be executed, which personnel is to execute the tests, and so on. In his planning the project manager usually does not go into the details of the requirements. In most cases, the level of details is such that specific requirements or subjects are assigned to each tester. The project manager records the results in the test plan. Test controlling can be conducted according to this document. No peculiarities for mathematical testing result from this phase of the test process.

---

2   See Peter Liggesmeyer: Software-Qualität - Testen, Analysieren und Verifizieren von Software. Spektrum Akademischer Verlag, Heidelberg/Berlin 2002

3   See Andreas Spillner und Tilo Linz, Basiswissen Softwaretest: Aus- und Weiterbildung zum Certified Tester - Foundation Level nach ISTQB-Standard, Dpunkt Verlag, August 2005.
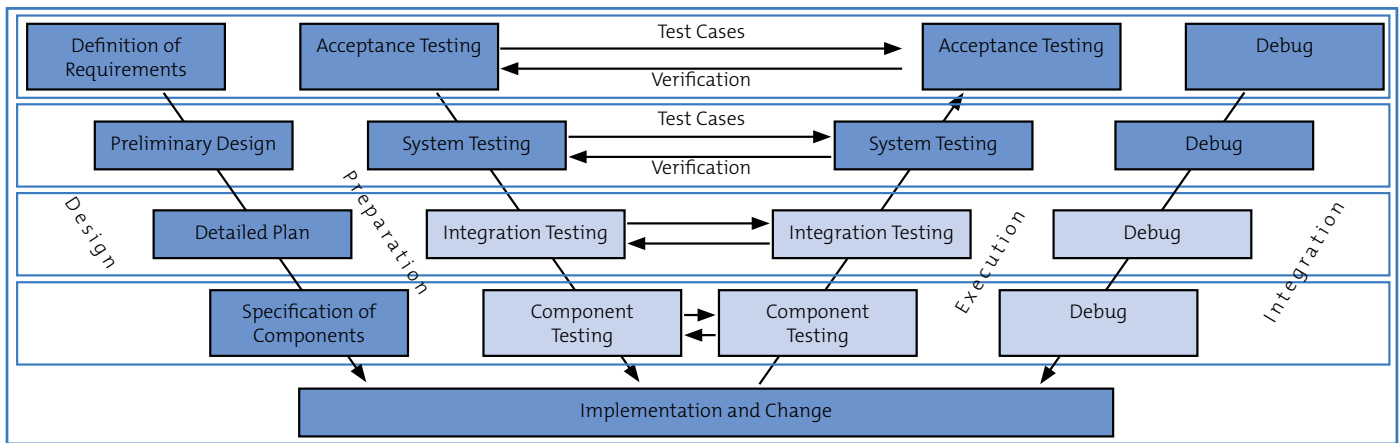
4   See IEEE 829

Figure 6: Levels of Mathematical Testing in the W-Model

2.b. Test Analysis and Design

In test analysis the basis for testing (requirements, specifications) is reviewed and analysed with regard to a sufficient level of detail for the creation of test cases. If necessary, the originator has to be consulted or amendments have to be made. Here, the testability is evaluated based on test basis and test object. Test specification is conducted according to the W-model (see Figure 6) if possible in parallel to the drawing up of requirements. This permits the early quality assurance of the requirements.

Once the analysis has been completed, logical test cases are defined (also called test design). Test cases can be specified based on the specification of test objects (black-box procedure) or on the program code (white-box procedure).

In mathematical module testing (component testing), white-box procedures like statement, branch or path coverage can be used, as can black-box procedures like equivalence partitioning. Testing level integration testing uses black-box procedures only. Use of white-box procedures does not make sense, because testers usually do not have access to the program code.

The creation of logical test cases – and the recording of the test design specifications specified according to IEEE 829 – is the first of two steps in the specification of test cases. The second step is part of the test process phase test implementation and documentation, which will be described later.

Various types or methods of testing can be applied when creating logical test cases. Among them are functional testing, which tests the visible input/output behaviour of the test object, structure-based testing, in which the internal structure of the test object is used for testing, and change-based testing, i.e., regression testing after changes to the system. Structure-based testing in a compound component like the mathematical sub-system requires a lot of effort, therefore requirement-based, functional testing is applied in both mathematical component testing and mathematical integration testing.

The requirements to be implemented are used as basis for testing. For each (new) functionality described in the requirements, at least one test case is derived. Test cases can be further refined using additional black-box testing methods like the boundary value analysis. If, based on experience in other projects, a tester assumes that a defect may occur with a particular combination, this test case should be included in the selection of test cases. This is also known as the experience-based approach.

In practice there also are requirements concerning certain subjects, in which the functionalities cannot be read directly from the requirements. Examples for this are new tariff generations or new profit sets. Here it is assumed that the already existing functionalities are valid for these requirements, too. Test cases have to be created carefully using equivalence partitioning. The greatest danger in this is defining too many test cases. To reduce the required effort, selected test cases can be complemented by

regression testing and by checking the production data.

When defining the expected results (test oracle), target data is derived from the input data using the specifications. In mathematical testing of portfolio management systems, test tools are often used as test oracles (back-to-back test). In this, a test tool (Excel, VBA, APL, or the like) is programmed according to the specifications used in the implementation of the portfolio management to compare the mathematical values of the portfolio management system. In case of deviations in the results from both versions after processing of the same input data, these deviations are analysed and the causes are resolved. In case of identical output, the test result is regarded as being correct. Identical defects in the diverse versions cannot be detected using this testing method.

The offer program can also be used as test oracle, because it has to be programmed anyway. It is, however, of limited use only, because usually programs to create offers can only test the issuing of policies for current tariff sets. Technical changes, inclusion/exclusion of tariffs and old tariff sets cannot be tested this way. Often a program to create offers uses the same calculation core as the portfolio management. In this case the use as test oracle is not an option. Manual calculation of the mathematical values of a portfolio management system from the specification often requires a lot of effort. Here, back-to-back testing provides a simple option to reduce the time required.

The test oracle (testing tool) should be realised early on during test design. A precise definition of the expected values is not necessary when using testing tools as test oracles. It is sufficient to complete the development of the tool before the test is executed ("post-calculation is easier than pre-calculation"). There are, however, cases in which it is very helpful for the developer to know the expected results. Most importantly this is the case when correcting test incident reports (defects reported by defect tracking).

Criteria for the successful processing of test cases are also part of the definition of logical test cases. A suitable metric to determine the classification level is, for example, the deviation between the calculated and expected value of an output parameter. These deviations can be percentages or absolute values. A possible level of classification in regard to absolute differences for acceptance is:

- < 1 Cent: acceptable

- 1 Cent – 1 EUR: conditionally acceptable. It has to be analysed whether the deviation is, e.g., a deviation between test oracle and portfolio management system due to rounding (e.g. rounding the insurance sum to EUR) or a systematic deviation.

- > 1 EUR: not acceptable

2.c. Test Implementation and Execution

Test implementation consists of preparatory tasks for the test. When the test process has proceeded and there is more clarity about the technical implementation, the logical test cases are

detailed and recorded in the test case specifications defined according to IEEE 829. These detailed test cases can then be used in testing without further changes or amendments if the defined preconditions are valid for the particular test case. The detailed course of testing is to be defined in the test procedure specification (see IEEE 829). In this the priority of the test cases has to be considered, and test cases have to be grouped in test sequences (also called test suites) and test scenarios (also called test scripts) to be able to effectively process the test cases and to get a clear structure of the test cases. In case a test has to be terminated prematurely, prioritising is to ensure the best possible result up to that point in time. For mathematical testing of a new tariff set, for example, issue of policy and update can be tested with a higher priority than technical changes.

Before starting the test, test beds have to be programmed and implemented in the environment concerned. The correct build of the test environment must be checked because failures can be caused by error conditions in the test bed. Also, the test oracles (testing tools) have to be programmed and must be executable.

After completing the preparatory tasks for the test, testing can be executed immediately once the parts of the system to be tested have been programmed and transferred. Testing can be executed manually or with tool support, and the predefined sequences and scenarios executed. In module testing, test beds and test drivers often allow to automate the tests.

First, it is checked whether the parts to be tested are complete. This is done by installing a test object in the available test environment and testing for general starting and execution capability. It is recommended to start with testing the main functionalities of the test object (smoke test). In case failures or deviations from the target results occur here already, it makes little sense to go deeper into testing.

While executing the test, the test cases processed and the results achieved (successful or with faults) have to be documented in the test log (see IEEE 829). Based on the test logs, the execution of the test must be traceable for persons that are not directly involved. It must be possible to verify that the planned test strategy was realized and to see which parts were tested when, by whom, in which detail and with which result.

Besides the test object, test beds, input files and test logs are part of each testing procedure. This information has to be managed in such a way that it is possible to repeat the test using the same data and frame conditions at a later point in time without any problems (configuration management of testware).

In case of a difference between actual and expected results during testing, the deviation has to be analysed, and it has to be determined whether there really is a failure, in which case a first check of possible causes has to be conducted. For this it may be necessary to specify and run additional test cases. In case deviations can be traced back to failures, this is documented in a test incident report (see IEEE 829). The causes for discrepancies to the target result can also be a faulty or inaccurate test specification, a faulty test infrastructure, a faulty test oracle (especially in second programming in testing tools) or a faulty test case or test run. Deviations have to be checked and analysed most carefully. A reported alleged failure which was caused by a faulty test case or faulty programming in the test oracle can damage the credibility of the tester. This must, however, not lead to potential failures not being reported just to be on the safe side. When documenting the deviations in a test incident report, it is important to ensure that a third person can understand the failure. The desired behaviour and expected values have to be indicated. In contrast to test design, in defect tracking it is not sufficient to indicate the testing tool used. The developer who has to correct the defect cannot be expected to deal with the testing tools.

Measurements of the testing coverage have to be conducted to monitor progress. This task is the responsibility of test management, because the individual testers will aim to execute all test cases.

Once the defect has been corrected, it must be checked whether the failure was removed and no new defects were added during the correction (regression testing). If necessary, new test cases have to be specified to check the program modules that were changed.

2.d. Evaluating Exit Criteria and Reporting

This phase evaluates whether the test completion criteria defined in planning have been met. This activity does not put special demands on mathematical testing. The tester should, however, support the project manager in clarifying discrepancies.

2.e. Test Closure Activities

In this phase, the experiences gained in the previous phases are analysed and the lessons learned are documented. This activity also does not put special demands on mathematical testing.
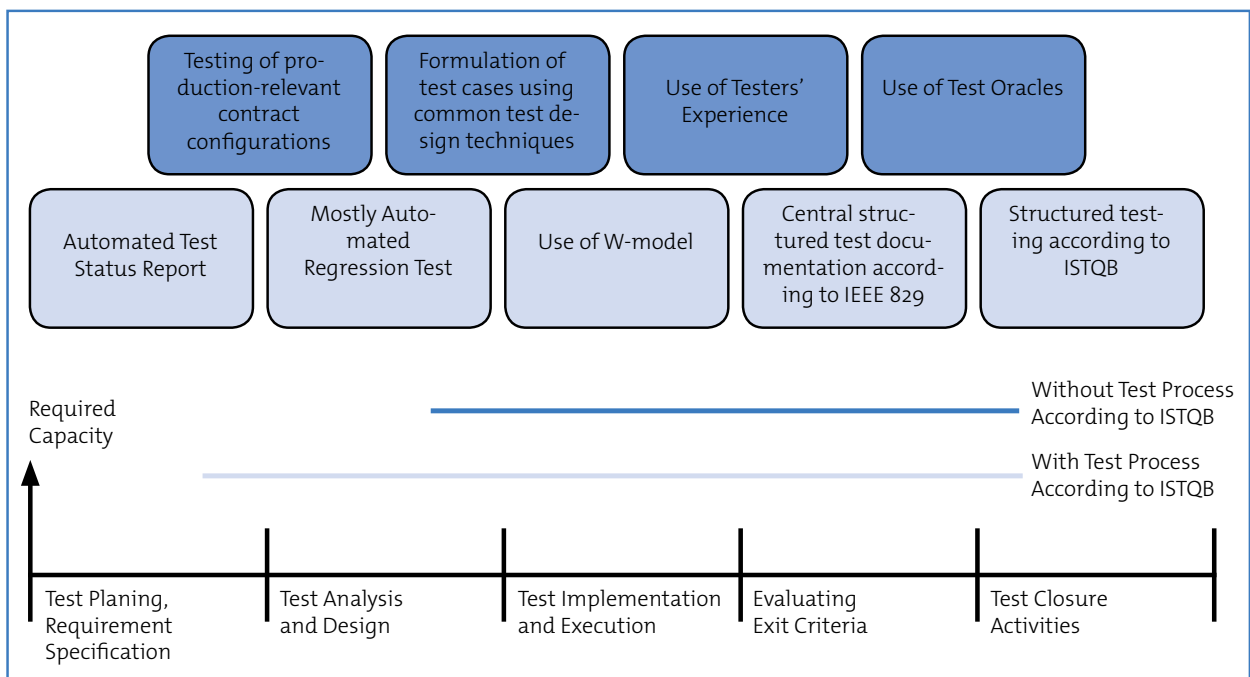

Figure 7: Test Efforts and Characteristics With and Without Test Process According to ISTQB

Conclusion

The development of test cases using common test techniques and the testing of production-relevant contract configurations are common test procedures in the insurance industry. Because of the complexity of the mathematical calculations, test oracles that can calculate the course of an insurance are also used frequently.

Unfortunately, in practice the subjects addressed above, like the structured test procedure and the use of the V-, or even better W-model, are seldom applied consistently. Often, there is no time for the test preparation tasks (analysis and design). Structured testing according to the ISTQB test process can improve this significantly. Also, test documentation is mostly not consistent and hardly suited for automatic evaluation. Consistent documentation can be achieved by test documentation according to standard IEEE 829. With it an automatic test status report can be created more easily.

The formal test procedure according to ISTQB does not require more time than the procedures used up to now. In the test process according to ISTQB, testing has to start earlier but requires less time, especially towards the end of the project. According to the W-model testers can start test design in parallel to the requirement specifications, thus ambiguities in the specifications are clarified early on. Because of this no, or at least no serious, need for clarification occurs during the actual testing. Also, when using this procedure the versions distributed by development are usually less defective. Thus fewer cases tested negative have to be corrected and tested again. Avoiding multiple runs thus positively affects the progress of testing.

During the development phase the tester has time to program the required test tools (test oracles, tools for regression testing, and the like), so that these are available at the first release of the development. Thus tests can already provide good results at the first release and be executed faster.

Figure 7 compares the efforts for testing using the formal test process according to ISTQB to those without using the formal test process.

A standardized test documentation makes it easier for testers to document their test cases. Using suitable tools the majority of the values to be documented can be automatically assigned. Thus the effort for documentation can be reduced, although more is being documented. Based on our experiences from projects, we can say that this documentation procedure guarantees auditing acceptability. In projects where we implemented the detailed and automated test documentation and test evaluation, this was praised by external and internal auditors.

Also, it must be considered that this test procedure in mathematical testing in the insurance industry requires qualified testers, who have acquired both the technical knowledge of the insurance products and the structured testing procedure.

According to our experience, the basic test process and the corresponding documentation according to the documents defined in standard IEEE 829 improves the quality in the mathematical testing of portfolio management systems significantly. Also, it is more cost-efficient, because production problems that are expensive to solve are avoided. This effect is multiplied with recurring activities, e.g. when several releases are introduced simultaneously.

Biography

Jens Fricke is a senior consultant at viadico ag. For more than 10 years he has been working as test analyst and test manager in several projects for implementation of portfolio management systems in the insurance industry. Jens received a diploma degree in mathematics (Dipl.-Math.) and the degree of an actuary DAV (German Actuarial Association). He is ISTQB Certified Tester and a member of ASQF e.V.

Thomas Niess is a consultant at viadico ag. He holds a degree in Economic Mathematics (Dipl.-Math. oec.) from the University of Ulm, with main focus on Finance and Actuarial Science. After his graduation he started his career at Finincon GmbH, Filderstadt, which in 2007 merged into viadico ag. He works as a consultant in mathematical testing of portfolio management systems in life insurance. Thomas is ISTQB Certified Tester and a member of ASQF e.V.