Titel, samenvatting en biografie

Remco Kwinkelenberg
Model-based Testing Enhances Action-word Based Testing to Boost Test Automation
Voorjaarsevent Testnet: 30 juni 2008

Samenvatting:

The growing complexity of software applications (business legacy applications, service oriented architectures, ERPs) and the necessity of retaining an overall view of software development (offshore) entail the implementation of high-performance application validation strategies. Functional testing represents a keystone for risk management, quality management and time-to-market constraints. Model-Based Testing is a solution that implements functional tests based on business requirements and test design automation (test cases, executable scripts, test plan coverage) while guaranteeing functional coverage completeness. This solution encompasses the processes, tools and best practices of the test center to improve conformity, traceability and risk management.

Model-based testing is often compared with action-word testing as two competitive test strategies. We show in this presentation that a complete application model-based testing (for automatic generation of automated tests) subsumes action-word testing, include both concepts into a smooth process to strengthen test generation and coverage.

The conjunction of model-based testing and action-word based testing for fully integrated test automation results in a great process improvement in a test center.

The action-word based testing methodology introduces the concept of action-words to split the design of test scripts into two deliveries: the test scripts on the one hand, the test driver on the other hand. .

Model-based testing does not aim at replacing action-word based testing: it simply completes it. The action-word concept is still valid. The test driver is still used. Only the conception of the test cases is automated and industrialized.

The tester no longer defines equivalence classes, test data and expected values manually. Instead, he uses the action-words defined by the analyst and sums them up into a model. As the analysis has already been performed when the action-words have been defined, building the model only consists in translating natural text into machine legible language, e.g. UML.

When model-based testing and action-word based testing are used together, then the model for test generation is only a graphical representation of the action-words, such as they have been defined by the analyst.

We show the relation between action-word and model-based testing on the basis of the LEIRIOS Smart Testing™ solution. The key features of LEIRIOS Smart Testing™ are:

– Monitoring of functional validation based on business requirements (Requirements Based Testing),
– Automatic generation of tests and executable scripts from functional modelling of the application,
– Set-up and organisation of specific testing-related competences within a service center (the test “factory”).

During test generation, LEIRIOS Smart Testing™ first checks the syntactical consistency of the model. Then, it generates the test states and the equivalence classes. Finally the test sequences (test step, input data and expected data) are generated to reach the previously identified test cases and to cover the equivalence classes. Doing this, LEIRIOS Smart Testing™ also checks the dynamical model consistency.

Thus, not only does LEIRIOS Smart Testing™ generate tests, but is also able to produce some quality check about the test model. As the latter results in the combination of the action-words, i.e. elements of the original specification after analysis, any error reported by LEIRIOS Smart Testing™ helps correcting either the specification and/or the set of action-words.

Finally, test cases are published as test scripts by means of a dedicated publisher. In this concept the scripts are a simple transcription of the test cases (i.e. operations and test data) into the desired target test environment (e.g. HP/Mercury, Borland, Compuware, IBM/Rational, etc…). The publisher additionally exports an interface file for the target environment containing the prototypes of operations defined in the model and invoked by the test scripts. Thus, not only does the modeling effort contribute to the test generation, but also to the model-driven development of the test driver on which the test scripts rely. Then automation engineers precisely know what stubs must be implemented, and what contract they must fulfill. This avoids the usual over-engineering of such tasks, which are often - wrongly - considered as off-process tasks.