

Automating Functional Test Design with Model-Based Testing

Stephan Schulz
Conformiq

ABSTRACT

Classic problems of functional software test development industry include often poorly documented coverage of specification requirements, challenging delivery cycles, high software quality requirements, costly test development, inflexible test sets, and high maintenance costs due to ever changing requirements. Technology and commercial software exist today that can help to resolve all of these issues by automating test design with model-based testing (MBT), i.e., by automatically generating tests and test documentation. In numerous case studies over a wide range of application areas MBT has been independently confirmed [9, 10] to cut testing costs significantly by increasing productivity, to enable of up to 91% code coverage during test execution [8], and to locate numerous defects that have escaped in conventional testing approaches. But deployment of MBT in organizations is a shift in paradigm, and requires a different way of thinking about test development and an adaptation of processes.

In this paper, we introduce automated test design and compare it with conventional software test development. We outline a methodology for developing functional tests from models that describe the operation of the system (aspect) to be tested. Although the approach is illustrated here with an AUTOSAR basic software module [7] example, it can also be applied in driver, system integration and even end-to-end testing. Automated test design is presented by showing modeling, steering of test generation, test case review, all the way to test script generation based on one of the commercially available MBT tools called Conformiq Designer.

CITATION: Schulz, S., "Automating Functional Test Design with Model-Based Testing," *SAE Int. J. Passeng. Cars - Electron. Electr. Syst.* 5(1):2012, doi:10.4271/2012-01-0010.

INTRODUCTION

Driven by technological advances and an ever-growing need for higher quality model-based testing (MBT), i.e., an umbrella of different approaches that use models in the context of testing, has matured in the last decade from a topic of research into an industrial technology also accepted in software and system testing. One approach is to use MBT for automating the design of functional software tests at various stages of product development including software component (e.g., CAN or MOST software components), software integration, and system integration or end-to-end testing. The availability of multiple general purpose and specialized commercial MBT tools and the emergence of first industrial standards on MBT [1, 2] attest this development. Across multiple application domains industrial deployments have increased coverage, produced more effective test sets, improved traceability and documentation, and drastically reduced efforts for test maintenance [9].

In the automotive domain or more generally in embedded system software development, use of MBT is particularly attractive as specifications, e.g., produced by AUTOSAR, already contain at least partial model information and modeling skills are common found in engineering staff.

ABOUT AUTOMATED TEST DESIGN

As mentioned in [3], the use of model-based testing for automated test design can be seen as the next step in the evolution of software testing after manual testing, test scripting, and test frameworks (including also keyword-based testing approaches). The pyramid in [Figure 1](#) symbolizes the efforts required to conduct and maintain tests for the same coverage of requirements. Note however that not all phases of testing have undergone these levels of automation. Especially when it comes to end-to-end testing many companies still today predominately use manual testing to validate system operation since at this level automation is very costly. But even here it has been shown that MBT can help to reduce costs [5] by optimizing test execution sequences so that overall test execution time or the amount of retesting features from earlier product development phases is minimized.



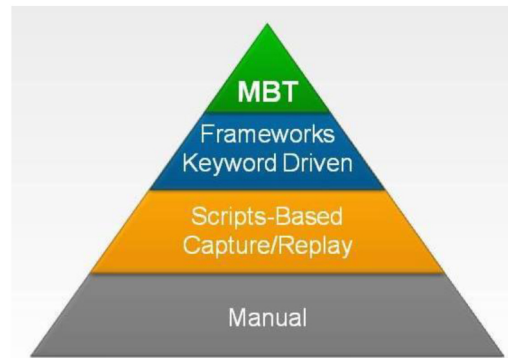


Figure 1. Evolution of Software Testing.

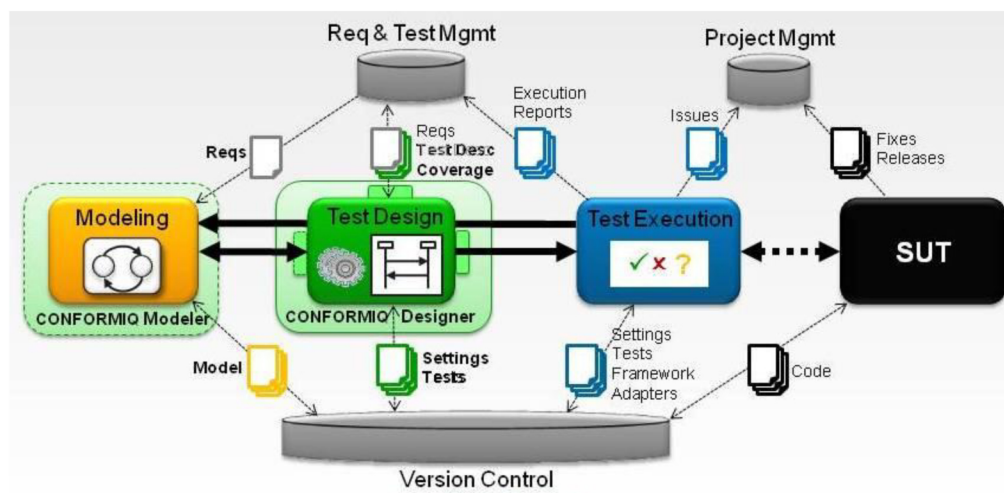


Figure 2. Software Testing Tool Landscape with Automated Test Design.

When it comes to the tool landscape involved in testing, MBT tools like Conformiq Designer integrate with existing tool chains, i.e., popular open source & commercial off the shelf version control, project, requirement, and test management solutions (such as DOORS, RQM, Quality Center), as well as test execution environments (such as ARUnit [6] or tools by Windriver, Elvior, Testing Technologies). Such infrastructure enables to trace from requirements throughout all test artifacts and vice versa, generating different test sets for specific purposes from the same source, rapid adaptation of test artifacts to specification changes, automatic generation of test documentation and requirement coverage reports, - essentially ultimate quality assurance. Deployments of MBT have been particularly successful in agile product development processes where constant change in the specification comes built into the process and engineers are forced to communicate more with all stakeholders. The availability of the system model and different presentation formats for generated test logic makes reviews with different stakeholders much more effective compared to review of test code or even handwritten tests. In addition, the time to change and maintain entire test sets is

drastically reduced since the model constitutes the source of all tests and one change to the model automatically updates the entire test set.

AUTOMATED TEST DESIGN IN A NUTSHELL

Automated test design refers to the automatic generation of tests from models that describe (an aspect of) the system functionality to be tested. What is needed to generate tests? Firstly, obviously we need to specify the model from which tests are to be generated. As in classic black box testing modeling starts with the specification of the system boundary, i.e., external interfaces available and accessible (either manually or automatically) for testing. After that operation or behavior of the system is specified, i.e., how the system to be tested is expected to process received stimuli and produce responses via these external interfaces. The behavioral part of the model may also include data manipulation, timing aspects, as well as further structuring of behavior into multiple parallel components. Important to notice is that the model needed for MBT should ideally not

be the design model. As in classic black box testing, models - like hand written tests - should be derived independently from the specification and not take design decisions into consideration. The objective is to test that our black box realizes the expected system operation.

Secondly, the user specifies coverage goals that our test set to be generated from the model should meet. Coverage criteria can be structural model criteria such as states, transitions, or 2-transitions but also conditions, requirements, boundary values, data value (combinations), etc. Note that this control of test generation is critical to use MBT effectively. The use of MBT does not change the overall testing problem: In general, only the execution of an infinite amount of tests can ensure that a system operates as desired. In addition, only a validated test is a meaningful test. Therefore, although at first glance a large number of tests may look very attractive, we realize quite fast that it is actually the opposite we want - since we have only a finite (and usually quite limited) amount of time available to validate and execute tests. Coverage goals coupled with smart test generation heuristics in MBT tools allow to reduce the usually a gigantic set of theoretically possible tests into a test set that is minimal (and even optimal) while at the same time complete in respect to user defined demands. In addition, traceability information allows users quickly to identify all the tests relevant, e.g., to a specific requirement.

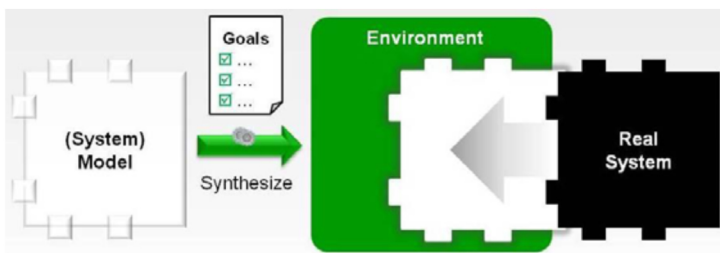


Figure 3. From MBT to Black Box Testing.

As shown conceptually in [Figure 3](#) MBT tools automatically generate or synthesize test logic in the form message flows with data and timing as well as the test oracle from model based on the user defined coverage goals, i.e., a set of traces through the system model behavior that match the selected coverage goals. This information can be generated even for complex models within a few minutes by exploiting multi core processors as well as use of formal verification techniques such as model checking and constraint solving. But test generation speed is only one part of the puzzle: The ability to comprehend automatically generated tests and the reasons why they were generated is another important factor. This requires MBT tools to provide users a variety of information to enable them to easily grasp the essence of a test or understand the difference to another test.

An example for a more concrete typical tool based workflow based on Conformiq Designer is illustrated in [Figure 4](#). The user composes a system model in a model editor, e.g., a simple statechart drawing tool like Conformiq

modeler or other popular UML tools like Rhapsody, Enterprise Architect or Rational Systems Architect. This model is then loaded into the MBT tool, e.g., Conformiq Designer, which can be considered the cockpit of the test developer. Here, coverage criteria are selected, test generation is triggered, tests & model are reviewed, coverage is analyzed, the model and/or coverage criteria are refined, or reviewed tests are transformed into documentation and/or test scripts for automatic test execution. Generated test scripts are text files specified which can be in one or more scripting languages that usually the test writer would specify in the test execution environment. Alternatively, e.g., in end-to-end testing, test documentation can be used as the basis for manual test execution.



Figure 4. Typical Workflow for Automated Test Design.

AUTOMOTIVE SOFTWARE TEST DEVELOPMENT WITH MBT

As discussed in the introduction the use of MBT requires a change in thinking about test development. [Figure 5](#) shows how a typical test development with MBT looks like. Most of these tasks are already practiced today in conventional test development. The new ones include model creation, extension, optimization as well as scripter creation, i.e., a MBT tool backend that renders generated tests into a specific graphical or textual format. Tasks such as interface abstraction and test review commonly happen also in conventional test development but often without formal recognition. Tasks for writing and maintaining actual tests and their documentation as well as the creation of test coverage reports are not in this figure since they are completely automated.

In the following sections we explain in a step wise manner how a model for test generation is developed, tests are generated and reviewed, and are then transformed into file formats for test execution based on an example from the automotive industry. Our example functionality to be tested here is the AUTOSAR ECU State Manager specified in [7]. This software component is located in the application layer in the AUTOSAR architecture as shown in [Figure 6](#). Please note that although this is an example of an application layer embedded software component, MBT can also be used for testing the operation of lower layer drivers as well as testing