



# Model-Based Testing

Ina Schieferdecker

Test consumes a large share of development efforts. If you don't continuously trim validation efforts and improve test efficiency while maintaining quality, your overall costs won't remain competitive. Model-based testing (MBT) strives to automatically and systematically generate test cases. In this column, Ina Schieferdecker introduces MBT technologies and methods. I look forward to hearing from both readers and prospective authors about this column and the technologies you want to know more about. —*Christof Ebert*



**IN MODEL-BASED TESTING** (MBT), manually selected algorithms automatically and systematically generate test cases from a set of models of the system under test or its environment. Whereas test automation replaces manual test execution with automated test scripts, MBT replaces manual test designs with automated test designs and test generation. However, a recent survey about the “Practice on Software Testing” shows that MBT hasn't yet arrived in industry, although its potential gains could be enormous.<sup>1</sup> Its prospective benefits include

- early and explicit specification, modeling, and review of system behavior, and early discovery of specification errors;
- better documentation of test cases, increased transparency, and enhanced communication between developers and testers;
- the ability to automatically generate useful tests and measure and optimize test coverage;
- the ability to evaluate and select regression test suites;

- easier updates of test models and suites for changed requirements and designs and for new product versions, and improved maintenance of test cases;
- higher test quality through model-based quality analysis; and
- shorter schedules and lower costs.

This column provides an overview of the state of the practice and MBT methods and tools.

## MBT Synopsis

First-generation MBT consisted of test generation from system models. This research developed the main concepts and algorithms for test behavior and test-data generation. However, it showed various shortcomings such as using a single model for code and test generation. This meant that errors in the model got propagated to the code and tests and were thus impossible to detect.

Second-generation MBT uses separate test models, which are sometimes called scenario models, usage models, environmental models, and so on. Test models support specifica-

tion of testing concerns in a dedicated model. This approach propagates the duality of the system and test system to the model level. This extends to the requirements level, which distinguishes between system requirements and test requirements and the models thereof.

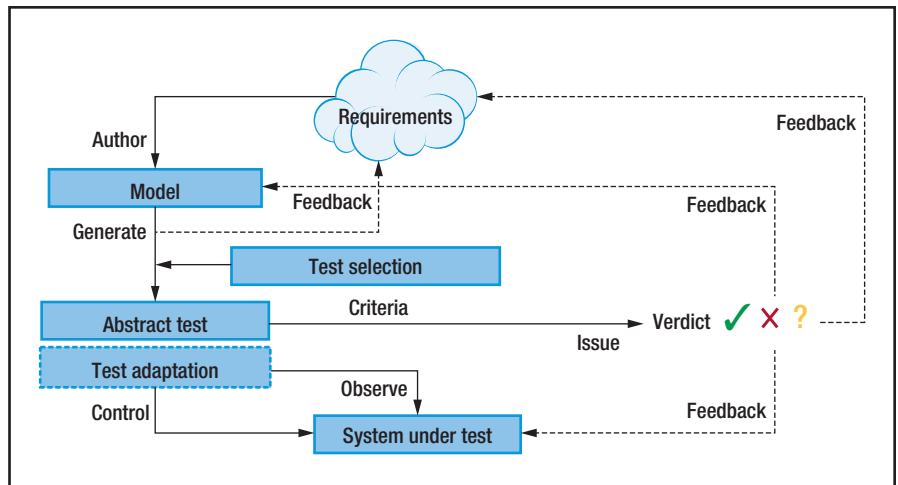
Second-generation MBT makes explicit the tester's expertise in denoting testing's essential concerns. Tests are designed by methods and tools established as industrial practice in model-driven software engineering and are documented during high-level review and discussion. The tests' quality—including correctness and coverage—can be predetermined at the model level.<sup>2</sup>

### MBT's Three Main Tasks

The generated tests can be executed manually or automatically, but combining automated test generation with automated test execution achieves the biggest gain. Key aspects of MBT encompass modeling notation and principles for test models, strategies, and algorithms to guide test generation and, optionally, on-the-fly generation or offline realization of executable tests. Figure 1 shows a typical MBT process in industry, which consists of the following three tasks.

#### Design a Functional Test Model

The test model represents the expected operational behavior of the system under test (SUT) or its environment. Test designers can use standard modeling languages such as UML to formalize the points of control and observation of the SUT, the system's expected dynamic behavior, the entities associated with the test, and test data for various test configurations. They link model elements such as states, transitions, and decisions to the requirements and later to the generated test cases and test results. The test models must be precise and complete enough to



**FIGURE 1.** Model-based testing (MBT) elements.<sup>3</sup> This process consists of three main steps: designing a functional test model, determining test generation criteria, and generating the tests.

allow automated derivation of tests from them.

#### Determine Test Generation Criteria

Models can usually generate an infinite number of tests, so test designers select criteria to limit the number of generated tests (for example, by selecting the highest-priority tests or ensuring specific coverage of system behaviors). A common approach for test selection is based on structural-model coverage (for example, determining the tests' coverage of model elements). Two examples of this approach are equivalence partitioning on the basis of the model's decisions or constraints and path coverage on the basis of  $k$ -pair transitions or selected execution sequences in the model.

Another useful type of criteria ensures that the generated test cases cover all the requirements, possibly with more tests generated for requirements with higher risk. In this way, testers can use model-based testing to implement requirements-oriented or a risk-driven test.

#### Generate the Tests

Typically, test generation in MBT is

fully automated. The generated test cases are sequences of high-level events or actions on or by the SUT, with input parameters, expected output parameters, and return values for each event or action. These sequences are similar to high-level test sequences that testers would design manually in keyword-driven testing (see Figure 1). Typically, the test sequences are easy to understand and complete enough for manual or automated test execution. If necessary, testers refine the tests to a more concrete level or adapt them to the SUT to support their automated execution.

#### MBT Tools

In the past, three commercial tools have led the MBT scene: Smartesting's CertifyIT, the Conformiq Tool Suite, and Microsoft's Spec Explorer. The next generation of more sophisticated MBT tools include Tedeso by Siemens/Imbus, Elvior's TestCast Generator, and All4Tec's MaTeLo. Many tools use TTCN-3 (Testing and Test Control Notation) as the language and execution technology for generated tests.<sup>4</sup> Table 1 compares these tools.

Companies are also deploying MBT tools for dedicated application domains.

General-purpose model-based-testing tools.

Tool	URL	Target domains	Test model	Test generation criteria	Test scripting
CertifyIT v5.1	<a href="http://www.smartesting.com">www.smartesting.com</a>	Software	Business Process Model and Notation or Unified Modeling Language (UML)	Test data and verification points	Textual test plans and executable test scripts in Quickset Professional and so on.
Conformiq Designer v4.4	<a href="http://www.conformiq.com">www.conformiq.com</a>	Data communications and telecommunications	State charts	Requirements-driven test generation, black-box test design heuristics	Textual test plans and executable test cases in Java, and so on
Spec Explorer 2010	<a href="http://research.microsoft.com/en-us/projects/specexplorer">research.microsoft.com/en-us/projects/specexplorer</a>	Software	Spec#	Transition coverage	Executable test cases in C# or on-the-fly testing
Tedeso 3.0	<a href="http://www.imbus.de/english/imbus-testbench/modules/managed-model-based-testing">www.imbus.de/english/imbus-testbench/modules/managed-model-based-testing</a>	Software	UML activity and sequence diagrams	Model and data coverage	Executable test cases in C++, and so on
TestCast Generator Beta	<a href="http://www.elvior.com/motes/generator">www.elvior.com/motes/generator</a>	Telecommunications, transportation, defense	UML state machines	State, transition, and decision coverage	Executable test cases in TTCN-3 (Testing and Test Control Notation)
MaTeLo 4.7.5	<a href="http://www.all4tec.net">www.all4tec.net</a>	Embedded systems	Enhanced Markov chains	Probabilities for transitions and inputs	Textual test plans and executable test cases in TTCN-3, and so on

For example, embedded-systems MBT tools include Reactis by Reactive Systems ([www.reactive-systems.com](http://www.reactive-systems.com)) and Modena by Berner & Mattner (<http://www.berner-mattner.com/en/berner-mattner-home/products/modena/index.html>). Research and open source MBT tools include Gotcha-TCBeans ([www.research.ibm.com/haifa/projects/verification/gtcb/index.html](http://www.research.ibm.com/haifa/projects/verification/gtcb/index.html)), GraphWalker ([www.graphwalker.org](http://www.graphwalker.org)), AutoFocus ([http://autofocus.in.tum.de/index.php/Main\\_Page](http://autofocus.in.tum.de/index.php/Main_Page)), Fokus!MBT ([www.fokus.fraunhofer.de/en/motion/ueber-motion/technologien/fokusmbt/index.html](http://www.fokus.fraunhofer.de/en/motion/ueber-motion/technologien/fokusmbt/index.html)), and Uppaal-CoVer ([www.hesselnu/CoVer](http://www.hesselnu/CoVer)).

### MBT in Standardization

A key issue in MBT is the plethora of concepts and methods that don't follow a common convention. Since 2005, the Object Management Group (OMG) has offered the UML Testing Profile (UTP) specification to support model-based

testing that's seamlessly integrated with UML. Today, several open source and commercial solutions implement UTP. Several research papers and companies (IBM, Microsoft, and so on) also reference it. OMG completed a UTP revision in July 2011.

Other standardization bodies have initiated approaches to a common nomenclature for testing. For example, several MBT tool vendors and major industrial users have developed a European Telecommunications Standards Institute (ETSI) standard to unify terminology and define a common set of concepts that MBT tools should support.<sup>3</sup> The MBT special interest group from the International Software Quality Institute has begun a training and qualification initiative ([www.isqi.org/en/modellbasiertes-testen-mbt.html](http://www.isqi.org/en/modellbasiertes-testen-mbt.html)). This technology-, tool-, and vendor-independent MBT qualification has four goals. The first is to let testers apply model-driven engineering methods

for test automation that are well established in software development. The second goal is to build the corresponding engineering skills. The third goal is to establish a qualification that serves as a standardized skill set for recruitment by providing a profound MBT certification rather than a number of specific tool qualifications. The final goal is to establish a qualification for career and training planning.

### MBT Rollout

MBT's successful application depends greatly on proper adaption to a company's development infrastructures and processes. One potential cause for disappointment is expecting too much from MBT. Companies often underestimate its rollout costs (see the "Model-Based-Testing Costs" sidebar). In addition, they often misclassify requirements for MBT infrastructure and personnel qualifications.

Studies on the application of MBT

recommend that companies introduce a rollout with several small pilot projects that usually cover a small field of expertise in the beginning. These projects should aim for simple, realistic, but well-attestable goals—for example, increased transparency, more efficient test specification procedures, and better test case maintenance. MBT experts and domain experts should propose, prepare, execute, and assess such projects to provide the necessary adaptation to the company and domain-specific requirements. A pilot project with low-hanging fruit should provide enough motivation to escalate MBT strategies to other projects, along with a suitable technological basis for such escalation.

Software development and quality assurance processes vary according to requirements, techniques, stakeholders, and target environments. Consequently, studies show that MBT is no commercial-of-the-shelf solution.<sup>5,6</sup> For successful implementation, MBT tools must be tailored to fit a company's processes and infrastructure. Such tailoring includes adaptation to the existing tool environment (test management tools, test execution environments, modeling tools, data repositories, and so on) and integration with best practices and existing processes.

### A Modular Tool Chain for MBT

The Fraunhofer Institute for Open Communication Systems (FOKUS) has developed Fokus!MBT ([www.fokus.fraunhofer.de/en/motion/ueber\\_motion/technologien/fokusmbt/index.html](http://www.fokus.fraunhofer.de/en/motion/ueber_motion/technologien/fokusmbt/index.html)), a flexible, extensible tool chain for MBT (see Figure 2). Fokus!MBT lets testers combine and integrate tooling as needed for MBT-based test processes.

Fokus!MBT facilitates automation of MBT processes for heterogeneous application domains. It's based on a service-oriented communication infrastructure of loosely coupled services



## MODEL-BASED-TESTING COSTS

When considering model-based testing (MBT), you should take into account the following costs.

### MBT TOOL COSTS

The cost of acquiring tools and frameworks to allow implementation is a necessary expense when switching to MBT.

### MODEL-DRIVEN ENGINEERING TOOL COSTS

Companies can couple implementation of MBT with implementation of model-driven engineering (MDE) processes. To fully exploit MBT's advantages, companies should have an MDE infrastructure (tools and methodology).

### ADAPTATION COSTS

The MBT methodology and tool platform must be fine-tuned with respect to the company's development processes, best practices, and domain requirements. Moreover, particular projects or project categories often require additional fine-tuning.

### QUALIFICATION COSTS

The implementation, integration, and maintenance of MBT procedures require a higher level of expertise than traditional test activities. Managers must consider the costs for qualification and training of current employees as well as for new experts.

### ROLLOUT COSTS

Changing existing methods, procedures, and best practices always involves rollout costs.

Requirement ID	Test case	Verdicts		
		Test case verdict	Local req. verdict	Global req. verdict
Req. 001	1	Fail	Fail	Fail
Req. 002	1	Fail	Fail	Fail
	2	Pass	Pass	
Req. 003	3	Inconclusive	Fail	Pass
	4	Pass	Pass	
	5	Pass	Pass	
Req. 004	5	Pass	Pass	Partial pass
	6	—	Not executed	

**FIGURE 2.** The Fokus!MBT approach. This tool chain for MBT lets testers combine and integrate tooling as needed for MBT-based test processes.

interoperating with each other in a distributed environment. It defines a proprietary testing metamodel to formally

represent test-specific information including the test model, requirements, logs, and so on. Telecommunications,

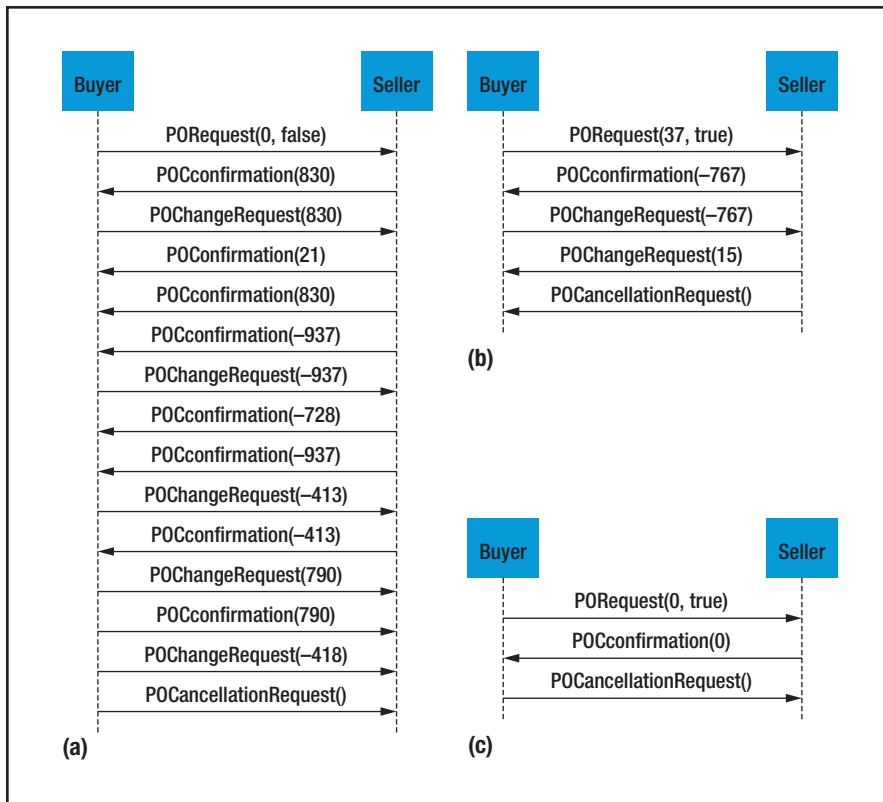


FIGURE 3. Generated test sequences and test results for FokusIMBT. (a) Random path, 60% transition coverage. (b) Shortest path, 60% transition coverage. (c) Shortest path, all states.

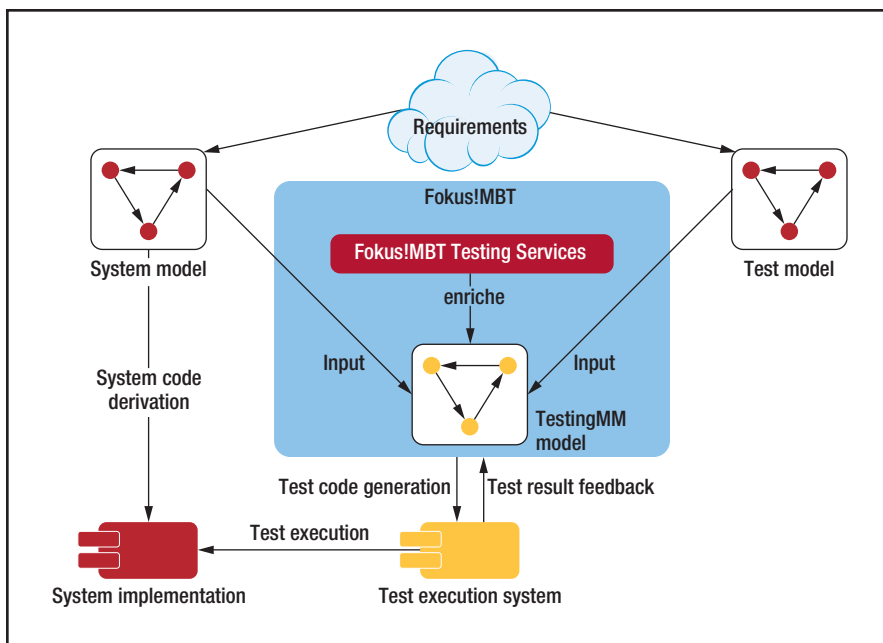



FIGURE 4. Model-based testing tool chain.

air traffic management, and safety-critical medical devices have successfully used it for model-based testing of software-based systems. Figure 3 shows some generated test sequences and test results created by using FokusIMBT. Figure 4 shows the MBT tool chain.

**D**riven by technological advances and the growing need for software quality, MBT has matured from a research topic to innovative leading-edge practices. It has succeeded in a range of domains, including information systems, embedded systems, communication networks, and distributed systems. Future developments will help automate the fine-tuning of MBT technologies and tools for domain-specific use in areas such as real-time or data-intensive systems. 

References

1. A. Spillner et al., “Wie wird in der Praxis getestet? Umfrage in Deutschland, Schweiz und Österreich,” *ObjektSpektrum*, May 2011 (in German); [www.sigs-datacom.de/fileadmin/user\\_upload/zeitschriften/os/2011/Testing/spillner\\_vosseberg\\_OS\\_testing\\_11.pdf](http://www.sigs-datacom.de/fileadmin/user_upload/zeitschriften/os/2011/Testing/spillner_vosseberg_OS_testing_11.pdf).
2. D-MINT (Deployment of Model-Based Technologies to Industrial Testing); [www.d-mint.org](http://www.d-mint.org).
3. *Methods for Testing & Specification (MTS); Model-Based Testing (MBT); Requirements for Modelling Notations*, ES 202 951 v 1.1.1, European Telecommunications Standards Inst., 2011.
4. *Methods for Testing & Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language*, ES ETSI ES 201 873-1 V4.3.1, European Telecommunications Standards Inst., 2011.
5. I.K. El-Far and J. A. Whittaker. “Model-Based Software Testing.” *Encyclopedia of Software Eng.*, J.J. Marciniak, ed., Wiley, 2001, pp. 825–837.
6. M. Shafique and Y. Labiche, *A Systematic Review of Model Based Testing Tool Support*, tech. report, SCE-10-04, Dept. of Systems and Computer Eng., Carleton Univ., 2010; [http://squall.sce.carleton.ca/pubs/tech\\_report/TR\\_SCE-10-04.pdf](http://squall.sce.carleton.ca/pubs/tech_report/TR_SCE-10-04.pdf).

INA SCHIEFERDECKER is the head of the Competence Center for Modeling and Testing at the Fraunhofer Institute for Open Communication Systems. Contact her at [ina.schieferdecker@fokus.fraunhofer.de](mailto:ina.schieferdecker@fokus.fraunhofer.de).