Model-Based Testing in Practice

Mark R. Blackburn, Robert D. Busser, Aaron M. Nauman, Travis R. Morgan
Systems and Software Consortium/T-VEC Technologies, Herndon, VA

Abstract: This paper discusses the use of model-based testing tools on a command and control application, and describes some key guidelines and benefits that were identified during the deployment of the model-based testing tools on this application.

1 Introduction

The integrated environment generically referred to as the Test Automation Framework (TAF) integrates government and commercially available model development and test generation tools. TAF integrates the DOORS® requirement management tool with the T-VEC Tabular Modeler (TTM) that supports the Software Cost Reduction (SCR) method [HJL96] for requirement modeling. DOORS integrates also with Simulink®, which supports design-based models, and TAF integrates requirement models with design models to provide full traceability from the requirements source to the generated tests, as reflected in Figure 1.

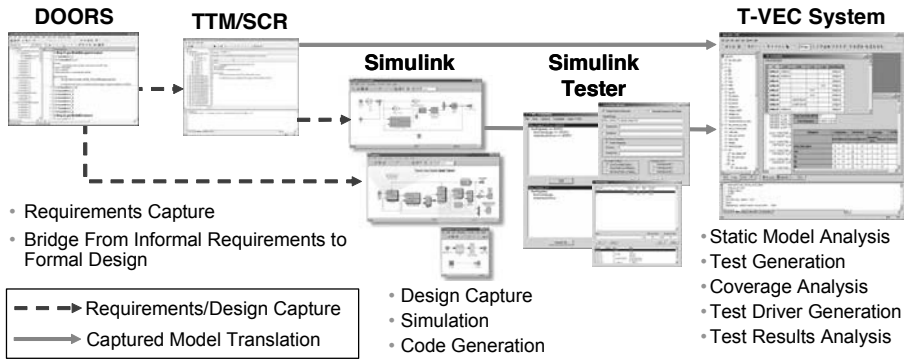

Figure 1. TAF Integrated Environment

Users have reported on the benefits of TAF's model-based testing support in terms of cost savings through test automation [St99, St00], early identification of requirement defects to reduce rework cost [Sa00], systematic support to identify critical system defects [BBKK01], advanced capabilities [BB04], and applicability to other domains such as security [CB03, BC04].

The paper discusses model-based testing tools and processes applied to a command and control application. The same basic processes have been applied to other applications within other application domains such databases, smart cards, language processing, client-server, distributed processing, avionics, and medical systems. The common process activities are applicable to software unit, integration, and system-level testing. Models typically describe the functional requirements of a system or component, although TAF has been applied to modeling security properties for a database [BBNC01]. The target implementations range from web-based to embedded systems on various platforms and operating systems (OS) and test drivers (aka test scripts) that were generated to support automated test execution in many languages and data formats.

2 Command and Control Monitoring Application

This company that produces the system described in this paper produces large, complex systems that often integrate with other large, complex systems. The application discussed in this section performs a monitoring function for many elements of an onboard command and control system. This, like many of the other systems, continues to be evolved, and there are often many interactions among the various systems. The requirement specifications for this element of the system span hundreds of pages. There are many related requirements, but conflicting requirements are difficult to identify by inspection because they often are packaged in different volumes with many versions. In addition, the testing process is manual and performed on a requirement-by-requirement basis, which again leaves conflicting requirements unidentified.

The MBT focused on software-system integration functionality that is tested manually. The objectives were to demonstrate the capabilities of the TAF tools and method and help this company do the following:

- Assess how they can formalize requirements using models
- Assess where test automation is feasible
- Construct and demonstrate a test automation framework tailored to their system and environment
- Estimate the return on investment (ROI) over the existing manual test process
- Learn how to adopt technology and tailor processes

3 Overview

The primary system component for the pilot project is called the Onboard Monitor (OM) element (Note: The names of the system elements have been changed to ensure anonymity

for this company). This element is part of a larger system called MASTER. There is requirement and interface documentation defined for this system, but important details, known by project personnel, were not in the documentation. The modeled requirements and associated tests primarily relate to messages received, processed, and transferred between various components of OM. Currently, most of the testing is manual and performed against a target, although there is a simulator called OMIPS that is used by the development organization to support test scripting using a language call Slang Script.

The OM system interfaces with several other system elements, as reflected in Figure 2. From a high-level point of view, the MASTER system is composed of the OM, and other elements that support tracking, decision logic processing for issuing commands, as well as the interfaces to the operator.

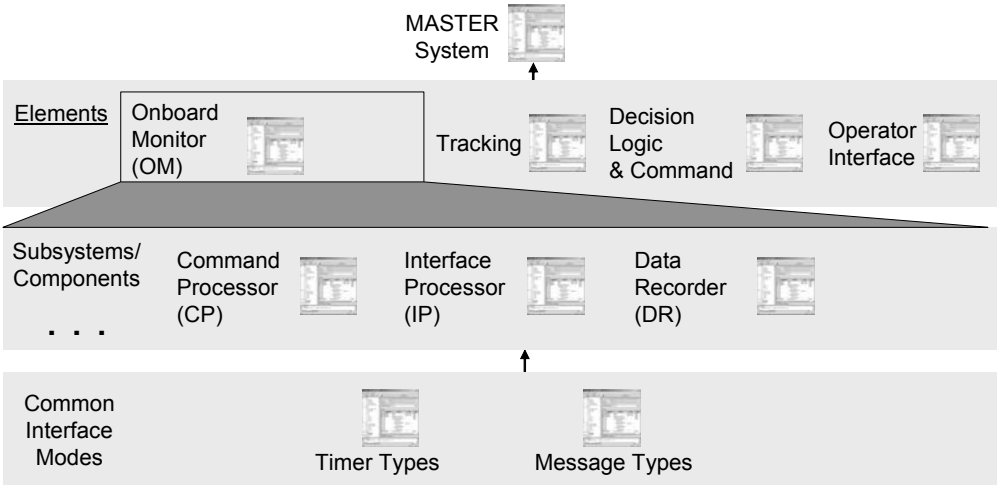

Figure 2. Model Hierarchy

Figure 3 provides a high-level perspective of the typical modeling process for the program. The objective is to use available requirement-related information, which comes in B-spec-like requirement documents, detailed interface specifications for the various messages, and some informal pictures that reflect analysis derived from requirement documentation and domain knowledge of the project engineers. Requirement models are specified in TTM, translated and T-VEC produces test vectors. The modeler, working with the requirement and design engineers, must correct requirement defects. Interface information is used to relate model variables to the actual system interfaces (API, message, etc.), and that supports automatic test driver generation of scripts that execute against a host, target or simulated system.

The combined company and TAF team (referred to as the MBT team) carried out the TAF evaluation over three different phases. During the first phase, the TAF team was successful in developing models for various requirements and scenarios that were allocated to the OM system. Working from the documentation and knowledge from the key engineers 67 requirement threads were modeled resulting in 121 test vectors during the first 2 days of the pilot project.

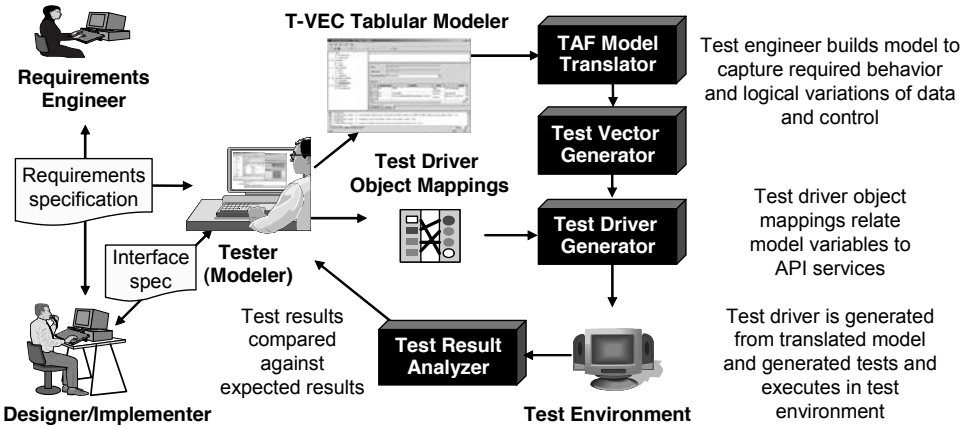

Figure 3. Model and Test Automation Overview

One of the modeled scenarios reflects the timed sequence interaction between two subsystems of the OM system (CP and IP), and the interaction to the OM. An example textual specification follows:

- If in the Start state (mode), the incoming message is 100, the prompt is YES, and the IP is UP, then the system should transition to the Prompt state.
- If in the Start state, the incoming message 104, the prompt is YES, and the IP is DOWN, then the system should transition to Failure state.
- If in the Prompt state, the incoming message is 111, the prompt is YES, and prompt_timeout < PROMPT_TIMEOUT (has not timed-out), then the system should transition to the Ready state.

After developing models for a number of these different scenarios, the team discovered that there were common models such as those reflected in Figure 2 called Timer Types and Message Type. TTM has model-include capabilities that allow a model of the requirements to be created once and then reused within other models for other related requirements.

Between the first and second pilot visits, the key modeler from the company developed a model that had 52 requirement threads with 77 unique test vectors. During the second phase,

the MBT team focused on developing test drivers for this model. The team developed a Slang Script that was executed through the OMIPS simulator, and produced actual results that were observable in the data recorder logs produced by the OM system. The pilot project helped in the following ways:

- Identified limitations in the OMIPS simulator; it could execute only about 20 test vectors per test script. The programmability of the test driver generator helped to overcome this limitation, the team modified the test driver schema to break large test driver script files into incremental files, each containing 15 test vectors.
- Identified some design for testability issues that need to be addressed with both OM system and OMIPS to provide more general support for test automation.
- OMIPS simulator developers and the OM system developer agreed to modify both systems in future versions to support greater testability. For example, one key change would increase test automation by providing GUI events issued using messages rather than by a manual interaction with the system; this would permit the OMIPS simulator to issue a program-generated message to the OM system without a human in the loop. Such commands could be generated by TAF.

The MBT team next focused on the final stages of the test automation process, including automated test execution, results analysis, and test report generation. The team created a test results analysis program that extracted actual outputs from the raw OM data recorder information. These data recorder records provide the actual test outputs that are compared against the expected outputs produced by TAF. The comparison is recorded in an HTML test report file. Figure 4 shows the conceptual environment.

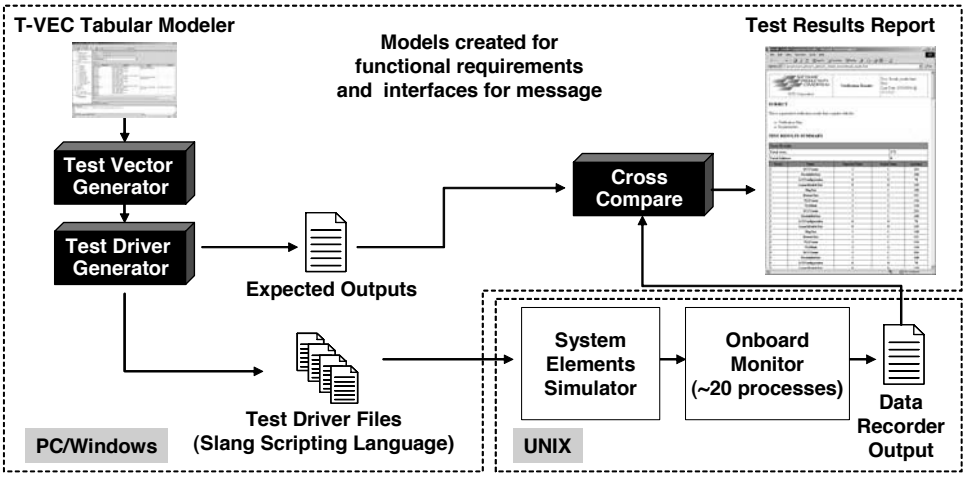

Figure 4. Fully Automated Test Automation Process Flow for OM Testing

4 Key Guidelines

Adopt a Technology Transition Plan. Companies should adopt a technology transition plan that grows the staff from the specialist developed in a prior project or pilot effort. The key modeler for this company demonstrated the skills of a model-based testing technologist, fulfilling the roles of both a modeler and test automation architect. He/she should be capable of carrying the recommended technology transition by leading one to six people in the use of the aforementioned process for a small set of requirements for the next release of some system baseline. The follow-on project then has additional team members to expand the team, where each person can mentor one to three additional people. When a company has a base of three to four project-experienced, model-based testing technologists, a larger group of 15 to 20 people can be trained to start a large project.

Use Model-Based Testing With Other Types of Testing. Model-based testing does not have to be all or nothing. Model-based testing can support a large percentage of the testing process, with a need to perform other types of testing. For example, there are two classes of messages processed by the OM system: solicited and unsolicited. According to this company's resident expert, at least 70% of the messages sent to OM are unsolicited messages, sent by other OM elements that must be processed by OM without additional communication with the other elements. Figure 4 shows a complete end-to-end test infrastructure and process for handling unsolicited messages.

Collaborate With Developers to Add Testability Support. For solicited messages, which are initiated by OM, usually through manual event, there is a need to enhance the testability of the current OM system in order to achieve complete test automation. External to the OM system, there is a need to have better controllability to simulate the external environment without manual intervention. It is also necessary to have some additional enhancements to the system for predictable observability of the test outputs. The team discussed the need, with the developer, to add a programmatic interface to the system that would allow external programs to stimulate system events such as solicited messages. The lead developer said it would be feasible and relatively inexpensive to implement the recommended programmatic interface in the next release.

Use Simulation for Early and Continuous Testing. Coincidently, the OMIPS simulation team was planning to develop a new simulator, and they were open to taking new requirements. Although there were a few limitations in the old OMIPS simulator, the MBT team's use of the simulator was valuable in providing requirements to further enhance their simulator. We were able to overcome the OMIPS limitation where the system cannot execute a Slang Scripts with more than about 20 tests. We modified the schema to produce multiple test driver files containing only 15 test cases each. We added a parameter that can be used to configure the test driver schema to put any number of test cases in a single test driver file.

Leverage System Internals for Analyzing Actual Test Outputs. Design for testability is key to test automation because it is important to be able to programmatically set inputs or initiate events, as well as obtain outputs that reflect the system behavior. The OM system receives thousands of messages but does not necessarily send out a response for each message; however, it does have a data recorder (i.e., internal logging) that collects all message inputs and associated process outputs. The project lead determined the raw data recorder file could be processed to extract actual outputs for automated test results analysis. The MBT team developed an OM test results comparison program in Perl to extract the actual outputs from the data record file and compare them with the expected outputs produced by the test vector generator, while producing an HTML test results report.

5 Results

The MBT pilot effort was successful in demonstrating an automated, end-to-end test process, as shown in Figure 4, that could be an order of magnitude more comprehensive than manual testing with 50% less time and cost. Prior to the pilot project, nearly all system and integration testing was a completely manual process. The pilot demonstration resulted in a few hundred test cases that represented several thousands of cases that are normally executed manually when the system capability is first developed, but then manually executed many times for each time the system is regression tested. The pilot project results are significant because the demonstrations reflected the feasibility to apply test automation to a significant base of the functionality of the system; for example the demonstrations were conducted using OM message requirements that represent 70% (or possibly more) of OM message processing, for which tests are currently performed manually. To carry out additional testing using this TAF process will require some minor changes to OMIPS and OM.

There are many other intangible ROI benefits. This process and the supporting test infrastructure used to support this pilot demonstration were 80% to 90% complete and relatively stable to support all follow-on testing. In addition, the team identified several requirements for the testing infrastructure that could further automate the process or change the underlying process for the organization. For example: Once an automated test suite exists, it can be run each time a build of the system occurs, allowing developers to identify bugs earlier in the process and making it easier to understand the specific changes that introduced a defect into the system rather than waiting weeks or months before manual testing is performed. The pilot project requirements and models were used to tailor the TAF training class exercise that were then delivered to company personnel that worked on different system elements.

Another important benefit that was observed during the requirement modeling process is that important requirement details often are not reflected in the requirement documents. Once the model is developed, these important details are captured (from domain experts). Related

requirements that often span different pages in a requirement document are captured in the same model. Companies often find that the captured models are the most valuable asset because they not only specify requirements in a non-ambiguous manner, but they are the source of the tests that can be generated systematically to provide complete test coverage from the requirements.

The first use of this technology requires some learning, but the first use by other companies indicates that even in the first use there is a significant increase in test coverage in less time than with existing manual processes. The key ROI gains should be obtained with each additional regression testing session that occurs. For this company, the time required for regression testing is essentially the same as it is to test the first time. With this automated process, the time to perform regression testing is easily less than 50% of the original time and cost and can be as little as 10%.

6 References

[BB04] Busser, R.D., L. Boden, Adding Natural Relationships to Simulink Models to Improve Automated Model-based Testing, Digital Avionics Systems Conference, October 2004. See http://www.software.org/pub/externalpapers/papers/busser-2004-2.doc.

[BBNC01] Blackburn, M.R., R.D. Busser, A.M. Nauman, R. Chandramouli, Model-based Approach to Security Test Automation, In Proceeding of Quality Week 2001, June 2001.

[BBKK01] Blackburn, M. R., R.D. Busser, R. Knickerbocker, R. Kasuda, Mars Polar Lander Fault Identification Using Model-based Testing, NASA Software Engineering Workshop, November 2001. See http://www.software.org/pub/taf/downloads/mars_polar_lander_2001.pdf.

[BC04] Blackburn, M. R., R. Chandramouli, Using Model-Based Testing to Assess Smart Card Interoperability Conformance, International Conference on Computing, Communications and Control Technologies, Austin, August 2004.

[CB03] Chandramouli, R., M. R. Blackburn, Model-based Automated Security Functional Testing , 7th Annual Workshop on Distributed Objects and Components Security (DOCSEC), Baltimore, MD, April 2003.

[HJL96] Heitmeyer, C., R. Jeffords, B. Labaw, Automated Consistency Checking of Requirements Specifications. ACM TOSEM, 5(3):231-261, 1996. See http://chacs.nrl.navy.mil/personnel/heitmeyer.html.

[Sa00] Safford, Ed, L. Test Automation Framework, State-based and Signal Flow Examples, Twelfth Annual Software Technology Conference, May 2000.

[St99] Statezni, David, Industrial Application of Model-Based Testing, 16th International Conference and Exposition on Testing Computer Software, June 1999.

[St00] Statezni, David. Test Automation Framework, State-based and Signal Flow Examples, Twelfth Annual Software Technology Conference, May 2000.