Quality of Test Specification by Application of Patterns

Justyna Zander-Nowicka
MOTION, Fraunhofer FOKUS
Kaiserin-Augusta-Allee 31
10589 Berlin, Germany
justyna.zander-nowicka@fokus.fraunhofer.de

Pieter J. Mosterman
The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098, USA
pieter.mosterman@mathworks.com

Ina Schieferdecker
MOTION, Fraunhofer FOKUS
Kaiserin-Augusta-Allee 31
10589 Berlin, Germany
ina.schieferdecker@fokus.fraunhofer.de

ABSTRACT

Embedded system and software testing requires sophisticated methods, which are nowadays frequently supported by application of test patterns. This eases the test development process and contributes to the reusability and maintainability of the test specification. However, it does not guarantee the proper level of quality and test coverage in different dimensions of the test specification.

In this paper the quality of the test is investigated and numerous metrics are defined. They are based mainly on the applied test patterns. They give a measure of quality for the test design and executed test cases with respect to a number of aspects. They also evidence the value of patterns application. If weighted, they enable to assess the executed tests.

Categories and Subject Descriptors

D.2.4 **[Software Engineering]**: Software/Program Verification – *Assertion checkers, Validation;*

D.2.5 **[Software Engineering]**: Testing and Debugging – *Monitoring, Symbolic execution*

General Terms

Design, Verification

Keywords

test quality, test pattern, online test, hybrid embedded software, software testing, Model-in-the-Loop, simulation, functional test

1. INTRODUCTION

Testing costs are well known to rise sharply, reaching at least 50% of software development time and effort [1, 2]. To reduce this cost and guarantee software quality, testing should be performed earlier in the software development cycle. Numerous efforts have contributed to making test activities more efficient, effective, and automatic.

So has our previous work [3, 4] where we presented a set of test patterns enabling the creation of an executable test model for hybrid embedded software. It has been called *Model-in-the-Loop for Embedded System Test (MiLEST)*.

In this paper we focus on the quality of those test patterns. By that, we deal with the problems of quality assessment for test specification and we calculate the achieved test coverage in terms of different aspects, which constitute the main novelties of this paper. Here, at least two dimensions are considered – the quality of a test design and test execution.

The contribution of this paper relates to the quality of tests that are defined by applying the test patterns.

- Hence, in Section 2, the MiLEST approach is introduced first.
- Then, we go through the test quality criteria available in the literature and select some of them for MiLEST (cf. Section 3.1).
- We investigate how the usage of patterns is influencing the quality of the test.
- We check how good the created test model is, defining the means to assess it. We handle the quality aspect of different test activities providing a classification of test quality (TQ) metrics for them. All that is introduced in Section 3.2.
- Section 4 gives a review of related work. Section 5 provides a case study to illustrate our methods. The final evaluation and conclusions complete the paper.

The quality of tests is quantified by application of the TQ metrics. Taking a test scenario that *every time a given threshold is exceeded, the flag should be changed*, we are able to assess how many and which variants of this scenario give a proper confidence level that the system under test (SUT) is behaving correct.

Although certain aspects of the test quality cannot be quantified without a context, the entire set of these aspects, if weighted, gives a measure of the test coverage w.r.t. different criteria.

We selected the MATLAB®/Simulink®/Stateflow® (ML/SL/SF) [5] to demonstrate the feasibility of our solution for *the test patterns* definition, their *application*, and their *quality assessment*. It provides a simulation engine that allows for the execution of tests, facilitating their dynamic analysis. This environment supports hybrid systems development and allows for using the same language for both system and test design [6].

2. MiLEST

At the early stage of system development, when its new functionality is introduced, its model serves as a primary instantiation of its realization. Hence, neither real-world nor reference signals are available for testing the model. This implies a need for another solution. In MiLEST we propose a new method for stimulation and evaluation of the embedded hybrid systems behavior. It is breaking the requirements down into the characteristics of particular *signal features (SigFs)*. For that purpose, a novel understanding of a signal is defined that enables describing it in an abstract way based on its properties (e.g., *decrease, constant, maximum*).

The MiLEST specifications reflect the structure of the system's requirements. The test development process and the abstractly aligned test system enable applying the concepts of *SigF generation* and *detection* mechanisms while building test specifications systematically. In this context, the *test system* is considered as a hierarchically leveled *test model* (also called *test design*).

Since the aim of the test system is not to provide the means for testing a single signal property but for validating complete SL/SF SUT models, independent of their complexity, structuring the test models in a proper way contributes to the scalability and reusability of the solution. Moreover, traceability of the test development artifacts is possible and transformation potentials emerge.

The structure of the test system consists of *four different levels* (see Figure 1) that can be built systematically and automatically. It makes the test system less error-prone leaving the test engineers plenty of scope for developing the complete test specification.

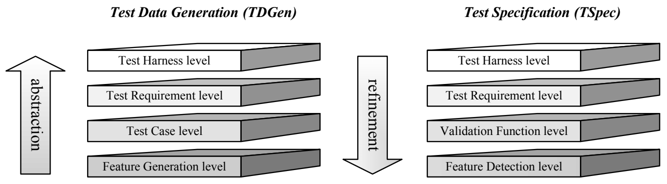

Figure 1. Hierarchical structure of the test system.

Technically, MiLEST is a SL add-on built on top of the ML engine. It consists of a SL library including test patterns, callback and transformation functions, and other ML scripts. The library is divided into: test specification, test data, and test control. Additionally, the test quality part includes metrics for assessing the quality of a given instance of a test model.

In Figure 2, a generic pattern for a test harness is presented. The test data (i.e., test signals used in the test cases) are generated within the test data generator shown on the left-hand side. The test specification, on the right-hand side, is constructed by analyzing the SUT functionality requirements and deriving the test objectives from them. It includes the abstract test scenarios, test evaluation algorithms, test oracle, and an arbitration mechanism.

In terms of the applied SL notation, the boxes represent the subsystems; the lines are carrying the signals.

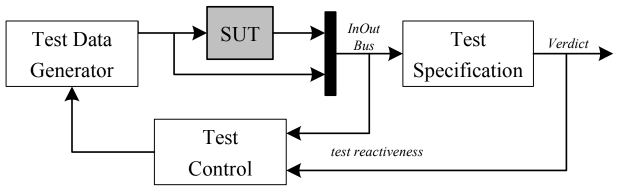

Figure 2. A test harness pattern.

The test specification is built by applying the test patterns available in the MiLEST library. Afterward, based on the already constructed parts of the test model, the test data generators are automatically derived. These are embedded in a dedicated test data structure. The automatic generation of test signal variants, their management, and their combination within a test case is also supported, analogous the synchronization of the obtained test stimuli. Finally, the SUT model excited with the previously created test data is executed and the evaluation unit supplies verdicts on the fly.

The first step in the test development process is to identify the test objectives based on the SUT requirements. For that purpose a high level pattern within the test specification unit is applied (see Figure 3). The number of test requirements can be chosen in the graphical user interface that updates the design and adjusts the structural changes of the test model.

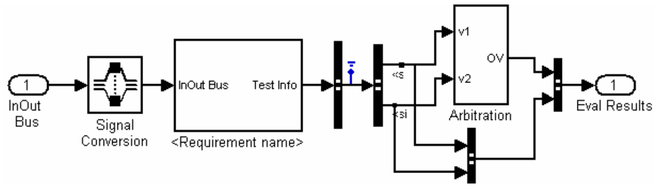

Figure 3. A test requirement specification pattern.

Next, validation functions (VFs) [3] are introduced to define the test scenarios, test evaluation, and test oracle in a systematic way (cf. Figure 1). VFs serve to evaluate the execution status of a test by assessing the SUT observations and/or additional characteristics/parameters of the SUT. A VF is created for any single requirement according to the conditional rules – *IF preconditions set THEN assertions set*.

A single informal requirement may imply multiple VFs. If this is the case, the arbitration algorithm accumulates the results of the combined IF-THEN rules and delivers an aggregate verdict. Predefined verdict values are *pass, fail, none,* and *error*. Retrieval of the local verdicts for a single VF is also possible.

Preconditions and assertions sets primarily include detectors for SigFs. VFs are defined to be independent of the currently applied test data. Thereby, they can set the verdict for all possible test data vectors and activate themselves (i.e., their assertions) only if the predefined preconditions are fulfilled. For all those elements patterns are defined. To illustrate, an abstract pattern for a VF (shown in Figure 4) consists of a preconditions block that activates the assertions block, where the comparison of actual and expected signal values occurs. A similar *cause-effect* analysis resulting in scenario patterns is discussed in [7].

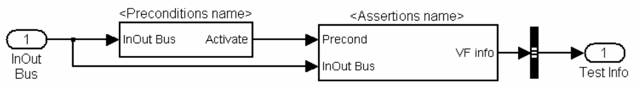

Figure 4. A validation function pattern.

A further step in the test development process is the derivation of the corresponding structures for test data sets and the concretization of the signal variants. The entire step related to test data generation is completely automatic by a merit of the application of transformations. Similarly as on the test specification side, the test requirements level for the test data is generated. This is possible because of information obtained from the previous phase.

Moreover, concrete SigF generators on predefined signals are created afterwards. These test signals are generated following a conditional rule of the form – *IF preconditions set THEN generations set*.

Knowing the SigFs appearing in the preconditions of a VF, the test data can be constructed from them. The preconditions typically depend on the SUT inputs; however they may also be related to the SUT outputs at some points in time. Every time a *SigF detector* is present for the assertion activation, a corresponding *SigF generator* is applied for the test data creation. Giving an example – for detection of an *increase* that is located in a precondition of a VF, a specific *signal increasing during a default time* is required.

The patterns for test data generation and concrete *SigF generators* are obtained as a result of the automatic transformations. Then, the test data variants are created. Though here, the necessary condition is a prior definition of the *signal value ranges* and *partition points* on all the stimuli signals. Equivalence partitioning and boundary value analysis are used in different combinations to produce the concrete variants for the stimuli.

3. TEST QUALITY BASED ON THE TEST PATTERNS

As far as any testing approach is considered, *test quality* is essential. It constitutes a measure for the test completeness and can be assessed on different levels, according to numerous criteria, applying metrics defined for them. In the upcoming section a few main categories resulting from the analysis of several efforts in the related work are distinguished. Then, the *TQ metrics* are defined for MiLEST. All of them are specified for the functional (i.e., black-box test related) test, leaving the structural (i.e., white-box test related) out of the scope.

3.1 Test Quality Criteria

Primarily, criteria similar as for software development are of importance. Hence, the consistency and correctness of the test development process and the resulting tests is considered.

Consistency of a test is defined as the degree of uniformity, standardization, and freedom from contradiction among the requirements documents, test design, test implementation, or test system. An example of the consistency check is *evaluating whether the test pattern applied in the test model includes a meaningful content*.

Correctness of a test is denoted in this paper by the degree to which a test is free from faults in its specification, design, and applied algorithms, as well as in returning the test verdicts. This definition is extended in comparison to *test correctness* provided by [8]. There a test specification is correct when it always returns correct test verdicts and when it has reachable end states. *Correctness of a test* can be exemplified in MiLEST when *it is checked whether the assertions set is complete enough to let the test pass*.

In this paper, *consistency and correctness of a test* are defined mainly w.r.t. to the test scenarios specified applying MiLEST patterns. Both of them can be assessed by application of the corresponding TQ metrics. Progress on the TQ metrics has been achieved by [9], where static and dynamic metrics are distinguished. The static metrics reveal the problems of the test specification before its execution, whereas the dynamic relate to the situation when the test specification is analyzed during its execution. An example of a static consistency check is evaluating *if at least one test for each requirement appears in the test specification*, whereas a dynamic check *determines whether a predefined number of test cases has been really executed for every requirement*.

Additionally, the authors of [8] define a TQ model as an adaptation of ISO/IEC 9126 [10] to the testing domain. There the characteristics are:

- *test reusability* and *maintainability* which are supported in MiLEST by the test patterns existence;
- *test effectivity* that describes the capability of the specified tests to fulfill a given test purpose;
- *reliability* that reveals the capability of a test specification to maintain a specific level of work and completion under different conditions;
- *usability* that describes the ease to actually instantiate or execute a test specification;
- *efficiency* and *portability* – left out of the scope in this paper.

3.2 Test Quality Metrics

For the purpose of this paper several *TQ metrics* have been defined for checking the efficiency and effectiveness of the test. They are ordered according to the test development phases supported by the MiLEST method. A more detailed version may be found in [11].

3.2.1 Test data related quality metrics

Signal range consistency is used to measure the consistency of a signal range with the constraints put on this range within the preconditions or assertions at the VF level. It applies to SUT inputs and outputs. The consistency for inputs is implicitly checked when the variants of the test signals are generated. In other words, the test data generator detects the inconsistencies in the signal ranges. The metric is used for positive testing.

Variants coverage for a SigF is used to measure the partitions coverage of a single SigF occurring in a test design. It is assessed based on the signal boundaries, equivalence partitions, and SigF type. The maximum number of variants for a selected SigF is equal to the sum of all possible meaningful variants. The metric can be calculated before the test execution by:

$$\text{Variants coverage for a SigF} = \\ = \frac{\text{\# of variants for a selected SigF applied in a test design}}{\text{\# of all possible variants for a selected SigF}}$$

Note that the sign # means "number of".

Variants coverage during test execution is used to measure whether all the variants specified in the test design are actually applied during the test execution. It returns the percentage of variants that have been exercised by a test. Additionally, it checks the *correctness* of the sequencing algorithm for the test data applied by the test system.

$$\text{Variants coverage during test execution} = \\ = \frac{\text{\# of variants applied during test execution}}{\text{\# of variants specified in a test design}}$$

Variants related preconditions coverage checks whether the preconditions have been active as many times as many different combinations of test signal variants stimulated the test. It is calculated during the test execution.

$$\text{Variants related preconditions coverage} = \\ = \frac{\text{\# of variant combinations present in a given test data set being applied during test execution}}{\text{\# of activations of a selected preconditions set}}$$

SUT output variants coverage is used to measure the range coverage of signals at the SUT output after the test execution. It is assessed based on the signal boundaries and equivalence partition points using similar methods as for the generation of test stimuli variants. The metric can be calculated by:

$$\text{SUT output range coverage} = \\ = \frac{\text{\# of the resulting variants for a selected output recognized after test execution}}{\text{\# of all possible variants for a selected output}}$$

3.2.2 Test specification related quality metrics

Test requirements coverage compares the number of test requirements covered by specified test cases to the number of test requirements contained in a corresponding requirements document calculated by:

$$\text{Test requirements coverage} = \frac{\text{\# of test requirements covered in a test design}}{\text{overall \# of test requirements}}$$

VFs activation coverage is used to measure the coverage of the VFs activations during the test execution. This metric is related to the *test requirements coverage*, but one level deeper in the MiLEST hierarchy. It is calculated as follows:

$$\text{VFs activation coverage} = \\ = \frac{\text{\# of VFs activated during test execution}}{\text{\# of all VFs present in a test design}}$$

3.2.3 Test control related quality metric

Test cases coverage is used to measure the coverage of the actual activations of test cases. The sequence of test cases to be activated is specified in the test control unit. The metric is calculated by the formula:

$$\text{Test cases coverage} = \frac{\text{\# of the activated test cases}}{\text{\# of all test cases present in a test control design}}$$

3.3 Realization

A few of the mentioned TQ metrics have been realized in the MiLEST. These are implemented either as SL subsystems or as ML functions. For instance, implementation of the *VFs activation coverage* is based on computing the number of local verdicts for which the value has been different from *none* or *error* in relation to the number of all VFs. The situation is illustrated in Figure 5.

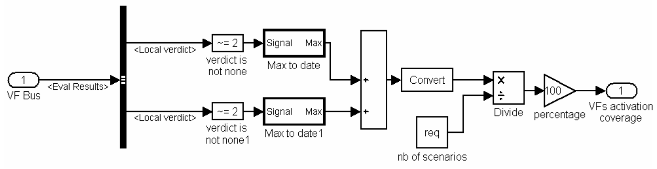

Figure 5. *VFs activation coverage* realization – exemplified for two VFs.

4. RELATED WORK

As [2] claims, metrics for structural test should be used in combination with those for functional test. For instance, the Model Coverage Tool in Simulink Verification and Validation [12] measures the *system model coverage* by collecting white-box information about model objects that have been executed.

For Time Partitioning Testing approach, in [13] – *cost/effort needed for constructing a test data set*, *relative number of found errors in relation to the number of test cases needed to find them* – are named as examples. These are the metrics that relate to the post-execution phase, not directly to the test patterns. Still, measuring the *cost* of pattern application is a very important factor determining the *gains* and *benefits* of the proposed methodology. In our case we obtain promising results as numerous steps within the test development are automated.

Apart from the measurements realized with the help of TQ metrics, consistency of the test specification may be checked statically by application of the test modeling guidelines using e.g., graph transformations [15] or Object Constraint Language [16].

Generally, the proposed pattern concept matches the definition given in [17]. The application domain is restricted to test engineering, though.

5. A CASE STUDY

This section demonstrates the application of the presented concepts for a selected case study. A simplified component – Pedal Interpretation (PI) of an Adaptive Cruise Controller developed by Daimler AG [2] is used. This subsystem can be employed as pre-processing component for various vehicle control systems. It interprets the current, normalized positions of acceleration and brake pedal (*phi_Acc*, *phi_Brake*) by using the actual vehicle speed (*v*) as desired torques for driving and brake (*T_des_Drive*, *T_des_Brake*) [2]. An example of a functional requirement is given in Table 1, while the SUT interfaces are presented in Table 2.

Table 1. Functional requirement for PI [2].

Normalized accelerator pedal position should be interpreted as desired driving torque – *T_des_Drive* [Nm]. The desired driving torque is scaled in the non-negative range in such a way that the higher the velocity is given, the lower driving torque is obtained.

Table 2. SUT inputs of PI.

SUT Input	Velocity	Acceleration pedal	Brake pedal
Value Range	<-10, 70>	<0, 100>	<0, 100>
Unit	m/s	%	%

5.1 The Test System

Let us analyze the requirement, which is realized by the pattern from Figure 3, for illustration purpose. It is interpreted as a set of IF-THEN rules. The VFs for them are designed as shown in Figure 6. As an example, the first VF is obtained from the following rule: *IF v is constant AND phi_Acc increases AND T_des_Drive is non-negative THEN T_des_Drive increases.* For further details please refer to [3, 4, 11].

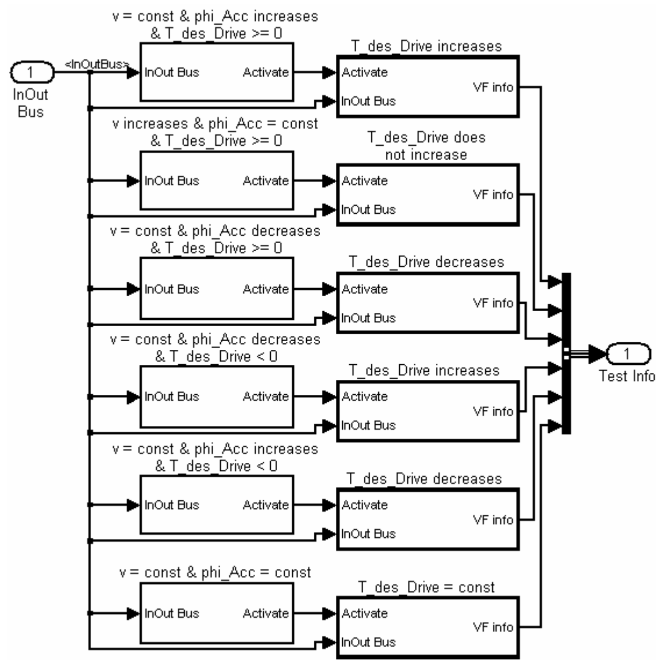
1
InOut Bus
<InOutBus>
v = const & phi_Acc increases & T_des_Drive >= 0
T_des_Drive increases
v increases & phi_Acc = const & T_des_Drive >= 0
T_des_Drive does not increase
v = const & phi_Acc decreases & T_des_Drive >= 0
T_des_Drive decreases
v = const & phi_Acc decreases & T_des_Drive < 0
T_des_Drive increases
v = const & phi_Acc increases & T_des_Drive < 0
T_des_Drive decreases
v = const & phi_Acc = const
T_des_Drive = const
InOut Bus
Activate
VF info
1
Test Info

Figure 6. VFs set for the requirement.

Then, the test data patterns are retrieved automatically from the test specification design. The test data generator (TDG) creates the representative test stimuli variants. The number of test data sets matches the number of VFs appearing in Figure 6.

Considering the first VF and the first test data set from Figure 6, the following applies: If the velocity is constant and an increase in the acceleration pedal position is detected then the assertion is activated. Thus, a constant signal for velocity is generated; its value is constrained by the velocity limits <-10, 70>. The partition point is 0. The TDG produces five variants from this specification. These belong to the set: {-10, 5, 0, 35, 70}. Furthermore, it is checked whether the driving torque is non-negative. It is the condition allowing the generation of the proper stimuli for the final test execution. For the acceleration pedal position limited by the range <0, 100> an *increase* feature is utilized. The details can be found in [4]. Evaluating this particular test scenario the following applies: If the driving torque increases as expected, a *pass* verdict is delivered, otherwise a *fail* verdict appears.

5.2 Test Quality Assessment

The simplest metric – *variants coverage for a SigF* – achieves 100% coverage if all possible variants of a given SigF, based on the selected criteria, are generated in the test design and applied during the test execution.

The *range* of a given signal must be consistent with the constraints given in the VFs' preconditions (i.e., such that the constrained values do not exceed the allowed range). *Signal range consistency coverage* checks this requirement in our approach.

Indeed, both of the metrics give the maximum coverage for our case study, the same as *VFs activation coverage*. *Test cases coverage*, however, reveals that only 80% of the designed test cases have been executed. This is because of the constraint put in the test control that *if all the VFs are activated at least once* (i.e., *VFs activation coverage* achieves 100%)*, the tests run should stop*. In consequence, similarly, *variants coverage during test execution*, *variants related preconditions coverage*, and *SUT output variants coverage* give 87%, 67%, and 59% coverage, respectively. When we eliminate the mentioned constraint and we execute the tests, maximum coverage values are achieved.

The experiment reveals the need to prioritize the applied metrics and weight their values relating them to each other. By that, the testing costs can be estimated.

6. SUMMARY

The *maximum* test coverage is an ideal case and it is usually not even attempted to be achieved (in contrast to the *maximized* one). The value depends mostly on the project costs and efforts spent on testing. Since the case study is relatively simple it serves for illustration purposes only. Nevertheless, it shows how the metrics relate to each other and how they can be weighted.

Moreover, the existence of test patterns gives a possibility to define the quantifiable TQ metrics related to these patterns (e.g., *VFs activation coverage*). The application of metrics, in turn, allows for evaluation of the quality of the test design and test execution as it has been illustrated for the case study. The lacking parts of test specifications are indicated.

MiLEST itself provides a set of patterns that support the systematic test design and enable the reuse of a number of structures for modeling the test. Additionally, some of the steps are automatic, which reduces the time of the test development.

7. REFERENCES

[1] Broy M., Jonsson B., Katoen J.-P., Leucker M., Pretschner A. (Eds.), *Model-Based Testing of Reactive Systems*, no. 3472 in LNCS, Springer-Verlag, 2005.

[2] Conrad M.: *Modell-basierter Test eingebetteter Software im Automobil: Auswahl und Beschreibung von Testszenarien*. PhD, TU Berlin, Wiesbaden (Germany), 2004 (in German).

[3] Zander-Nowicka J., Schieferdecker I., Marrero Pérez A.: Automotive Validation Functions for On-line Test Evaluation of Hybrid Real-time Systems", In *IEEE 41st AutoTestCon*, pp. 799-805, ISBN 1-4244-0052-X, 2006.

[4] Zander-Nowicka, J., Xiong, X., Schieferdecker, I.: Systematic Test Data Generation for Embedded Software. In *Proceedings of SERP 2008,* pp.164-170. CSREA Press, 2008.

[5] The MathWorks™, *MATLAB®/Simulink®/Stateflow®*, www.mathworks.com/products/matlab/.

[6] Schäuffele J., Zurawka T.: *Automotive Software Engineering*, ISBN: 3528110406. Vieweg, 2006.

[7] Tsai W.-T., Yu L., Zhu F., Paul R.: Rapid embedded system testing using verification patterns, In *IEEE Software*, Volume 22, Issue 4, pp. 68-75, 2005.

[8] Zeiss, B., Vega, D., Schieferdecker, I., Neukirchen, H., Grabowski, J.: Applying the ISO 9126 quality model to test specifications – exemplified for TTCN-3 test specifications. In *Proceedings SE 2007*, pp. 231-244. GI-LNI, 2007.

[9] Vega, D., Schieferdecker, I.: Towards quality of TTCN-3 tests. In *Proceedings of SAM 2006*.

[10] ISO/IEC Standard No. 9126: Software engineering – Product quality; Parts 1–4. Geneva, Switzerland, 2001-2004.

[11] Zander-Nowicka, J.: *Model-based Testing of Real-Time Embedded Systems in the Automotive Domain*, PhD thesis, TU Berlin, submitted in 2008.

[12] The MathWorks™, *Simulink® Verification and Validation™*.

[13] Lehmann E.: *Time Partition Testing, Systematischer Test des kontinuierlichen Verhaltens von eingebetteten Systemen*, PhD, TU Berlin, 2003 (in German).

[14] Grimm, K.: *Systematisches Testen von Software. Eine neue Methode und eine effektive Teststrategie*. PhD thesis, TU Berlin, ISBN: 3-486-23547-8. 1995 (in German).

[15] Amelunxen, C., Legros, E., Schürr, A., Stürmer, I.: Checking and Enforcement of Modeling Guidelines with Graph Transformations. In *Proceedings of AGTIVE* 2007, pp. 361-375. 2007.

[16] Object Constraint Language, version 2.0, May 2006, http://www.omg.org/docs/formal/06-05-01.pdf [08/26/08].

[17] Coplien, J.: Software Design Patterns: Common Questions and Answers, *The Patterns Handbook:Techniques, Strategies, and Applications*. Rising, L. (Ed.), 1998.