

How to Use Automatic Test Vector Generation for Model Coverage?

Sadegh Sadeghipour

IT Power Consultants
Gustav-Meyer-Allee 25
13355 Berlin / Germany
sadegh.sadeghipour@itpower.de

Abstract: Tools for the automatic generation of test vectors covering the model structural elements have already found their way into the industrial practice. The main question from the point of view of test engineer and quality assurance expert is the kind of the usage of these tools within the model-based test process to reach a maximized benefit. This paper describes different use scenarios of deployment of automatic test vector generation on model level to increase the efficiency and improve the quality of the test process.

1 Introduction

Coverage metrics are a measure of the effectiveness of testing; they can be used to control test depth and test termination as well as to point out coverage holes in given test suites. The determination of structural coverage metrics at code level (code coverage) as a part of the test process is considered to be best practice in industrial software development and is required by many development standards (see e.g. [8]). In contrast, structural coverage metrics at model level (model coverage) are already being introduced into the industrial practice [2].

Within the field of automotive control software development the users of Simulink/ Stateflow, which is widely deployed within the automotive industry to model the software of electronic control units, have been provided with a tool for measuring structural model coverage during testing by The Mathworks, Inc. a few years ago [10]. Also tools for the automatic generation of test vectors covering the model structural elements have emerged on the market of model-based CASE tools [1] [4] [9]. The tool Reactis Tester [9], which generates test vectors from Simulink/Stateflow models, has already found its way into the industrial practice and is used by the developers and testers of automotive control software.

A main question concerning the deployment of automatic test vector generation is the problem of test evaluation, i.e. whether or not the tests executed with the generated test vectors are successful (the oracle problem). In order to assess the tests one has to functionally interpret the automatically generated test vectors. However, this is a difficult task in general. Therefore, the main question from the point of view of test engineer and quality

assurance expert concerning automatic test vector generation is the following one: Does the deployment of automatic test vector generators bring any benefit to the model-based test process? Or, more precisely: How should an automatic test vector generator be used within the model-based test process to reach a maximized benefit? This paper describes different use scenarios of deployment of automatic test vector generation on model level to increase the efficiency and improve the quality of the test process.

2 Using Automatic Test Vector Generation within an Effective Test Strategy

A test strategy is effective if the tests included by that strategy are likely to reveal bugs in the tested object [3]. In order to increase the probability of revealing bugs a suitable combination of individual test approaches is generally suggested as an effective test strategy. A well-known effective test strategy within the conventional software development is the combination of functional and structural test approach [7]: At the first step functional (black-box) test cases are derived from the software requirements. During functional test execution the code coverage, concerning a pre-defined coverage criterion, is measured. At the second step new test cases are determined to cover the code elements, e.g. statements, branches or paths, not yet covered. This effective test strategy has been adapted for model-based testing [5]. According to it after defining functional test cases an appropriate model coverage criterion is determined and the model coverage is measured during the functional test execution. At the next step new test cases are determined to cover the model elements not yet covered. Exactly at this step an automatic test vector generator can be used to cover the model elements not covered so far (Figure 1). This saves time and effort needed for manual search of test vectors. Moreover, the functional success of the generated tests, and consequently their functional interpretation, is not the main concern, because the functional interesting tests have already been defined and executed at the first step. The automatically generated structural test cases can be assessed as successful, if no apparent error, e.g. overflows or deadlocks, occurs during the model test.

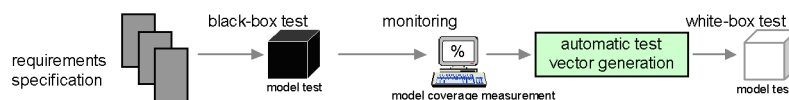


Figure 1: Effective model-based test strategy

3 Using Automatic Test Vector Generation for Back-to-Back and Regression Tests

Back-to-back tests establish the equivalence of different representations of the test object, e.g. between the model and the program code generated. Regression tests ensure that modifications made to the test object respect the required functionalities. When doing this, the approved results from former tests serve as reference data.

In these tests the tester is aimed at a proof of (partial) equivalence between the test object and a former approved version of it. Consequently, the test cases included in the required test suite do not need to be individually interpreted. However, the test suite must be sufficiently thorough and comprehensive to justify a claimed (partial) equivalence between the test object and the reference object in the case of successful tests. Therefore, an automatically generated test suite to cover the structural elements of the model would be a suitable candidate for back-to-back and regression tests (Figure 2). While the thoroughness and the depth of test is guaranteed by reaching structural coverage criteria, the effort to derive test cases is minimized.

Evaluation of back-to-back and regression tests, namely the comparison of outputs of the test object with the approved reference outputs, can also be automated. For an example see [6].

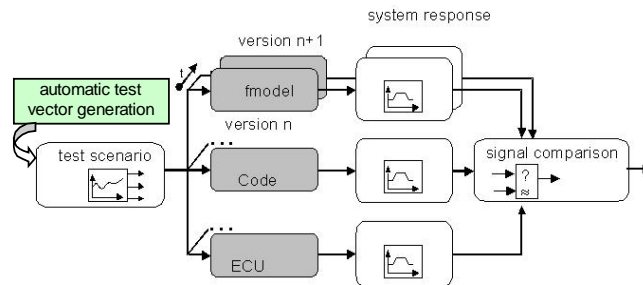


Figure 2: Using automatic test vector generation for back-to-back and regression tests

4 Using Automatic Test Vector Generation for Developer Tests

Within a bottom-up development strategy model developers would like to immediately test the small testable modules they create. Such modules correspond to individual algorithms or functions of the whole software to be developed. However, due to restricted time and close delivery dates they often waive such 'developer tests' and inevitably accept a low quality of the developed part of software. The possible errors made on this level may then be revealed later during module or system testing causing higher costs for their elimination.

Automatic test vector generators can assist the developers to generate test vectors for small testable units. Since the developers are best familiar with the structure of the modules they create, they can easily interpret the automatically generated test vectors and reliably assess the corresponding tests.

5 Using Automatic Test Vector Generation for Functional Tests

Last but not least automatic test vector generation can also be used for functional tests based on the requirements specification of the software. Indeed, this would only be possible if the test vector generator provides the user with a facility for modelling the test behaviour as a model running parallel to the test object. The user of Reactis for example is given such a possibility, called 'test objective' [9]. Reactis then automatically generates test vectors covering the structural elements of the test objective. Of course, in this case the test engineer has to invest some intellectual work to implement the test objectives from the requirements specification. However, this effort is inherent to functional testing and cannot be eliminated at all. The next step, i.e. definition of test vectors corresponding to the specified test objective, which is tedious if manually accomplished, is taken over by the test vector generator and needs no effort from the test engineer.

References

- [1] ATG tool (product information), OSC - Embedded Systems AG, <http://www.osc-es.de/products/en/atg.php>.
- [2] Baresel, A., Conrad, M., Sadeghipour, S., Wegener, J.: The Interplay between Model Coverage and Code Coverage, 11. Europ. Int. Conf. on Software Testing, Analysis and Review (EuroSTAR 03), Amsterdam, NL, 2003.
- [3] Beizer, B.: Black-Box Testing - Techniques for Functional Testing of Software and Systems, John Wiley & Sons, New York, ISBN 0-471-12094-4, 1995.
- [4] Conformiq Test Generator (product information), Verifysoft Technology GmbH, http://www.verifysoft.com/en_conformiq_testgenerator.html.
- [5] Conrad, M.: Modell-basierter Test eingebetteter Software im Automobil - Auswahl und Beschreibung von Testszenerarien, PhD thesis, Deutscher Universitäts-Verlag, Wiesbaden, ISBN 3-8244-2188-7, 2004.

- [6] Conrad, M., Sadeghipour, S., Wiesbrock, H.-W.: Automatic Evaluation of ECU Software Tests, to appear in SAE 2005 World Congress, Detroit, USA, April 2005.
- [7] Grimm, K.: Systematisches Testen von Software - Eine neue Methode und eine effektive Teststrategie. Dissertation, GMD-Bericht Nr. 251, R. Oldenburg Verlag, Munich, 1995.
- [8] ISO/TR 15497:2000 Road vehicles - Development guidelines for vehicle based software. International Organization for Standardization (ISO), 2000.
- [9] Reactis Tester (product information), Reactive Systems Inc., www.reactive-systems.com.
- [10] Simulink Verification and Validation 1.0.1 (product information). The MathWorks Inc., <http://www.mathworks.com/products/simverification>