Model-Based Testing in Practice

Alexander Pretschner

Information Security, ETH Zürich, 8092 Zürich, Switzerland

1 Introduction

Testing comprises activities that aim at showing that the intended and actual behaviors of a system differ, or at gaining confidence that they do not. The goal of testing is failure detection: observable differences between the behaviors of implementation and specification. Classical estimates relate one half of the overall development effort to testing. Even if Fagan [1] suspects that this percentage includes activities such as finding the causes of failures in the code and removing them, testing is an important and expensive activity in the development process.

Model-based testing (MBT) relies on models (specifications) that encode the intended behavior of a system. Runs of the model are interpreted as *test cases* (in this paper, *tests* for short) for a system under test (SUT): input and expected output. Activities in MBT have attracted a major interest in the past years. In addition to the appeal of the concept, we see the major reasons (a) in a gain of momentum of model-based languages and technologies (UML, MDA) and their seemingly direct connection to testing activities, (b) in the increasing popularity of test-centered development processes such as TDD or XP, and (c) in the possibility of promoting research activities and results under the umbrella of "lightweight" formal methods. Yet, despite numerous efforts in the area, it is not clear if the use of this technology pays off and, if so, in which niches.

This overview paper summarizes the ideas, promises, assumptions, and evidence of the benefits of MBT in Sec. 2. Sec. 3 argues for empirical studies.

2 Model-Based Testing

We describe the fundamentals of MBT, convey the appeal of the concept, pin down assumptions for successful deployment, and report on evidence.

Fundamentals. Testing consists of three stages: generation of tests, execution of tests, and derivation of verdicts. When applied to non-deterministic systems, the first two stages are likely to collapse. Models are used for the generation of tests and test harnesses. Since tests already comprise input and expected output we do not need models for assigning verdicts. Test harnesses are pieces of code that actually execute tests. Models that can be used to derive test harnesses are usually of a structural nature. These are not the subject of this paper; we will focus on behavior models instead.

The number of possible tests is normally very large or even infinite. For economical reasons, it should be reduced to a minimum. For quality reasons, it must be sufficiently high as to reduce the number of remaining failures in the field to an acceptable number. This means that test case generation, be it model-based or not, must face the problem of selecting "good" tests in the following sense. They are cheap to derive and cheap to execute; they are cheap to evaluate in that they help detecting faults on the grounds of failures; they find all "serious" and "frequent" failures.

Respective selection criteria can be divided into structural, functional, fault-based, and stochastic criteria. *Structural criteria* require that, upon execution of the SUT or the model, a certain coverage of data domains and of the nodes and edges in control flow and data flow graphs be obtained. Their ability to detect failures is subject to ongoing disputes [2], and particularly so when compared to random testing. On the other hand, structural criteria lend themselves to the automated generation of tests (test selection criteria are also quality indicators [3]), and they are measurable. This is relevant from a management point of view. *Functional criteria* try to capture isolated functionalities or requirements of a system, and define tests accordingly. This is usually achieved by defining models of the environment, or by other dedicated constraints on the set of all executions of the model. In general, MBT is hence concerned with models of both the SUT and the environment. Alleviating the task of defining concrete functional criteria seems to be possible from a methodological perspective, but not from a technical point of view. *Fault-based criteria* rely on knowledge of typically occurring failures. Finally, *stochastic criteria* rely on input probability distributions. If the distribution is uniform, then testing is random. Other distributions are based on user profiles [4] which is particularly appealing from an economic point of view since it caters for the frequency of potential failures.

Given a model and an adequately operationalized test selection criterion—constraints or environment models—a test case generator, human or automatic, then derives traces of the model, i.e. tests for the SUT. Test case generators use the technologies of model checkers, symbolic execution, satisfiability checkers, or deductive theorem provers. In addition to the practical difficulties with verifying large systems, model checking or theorem proving of the programs alone are not sufficient because these activities cannot transcend the assumptions on the environment (hardware, operating systems, legacy systems).

Methodologically, MBT makes sense only if the model is more abstract than the SUT. Otherwise, the effort of validating the model would exactly match the effort of validating the SUT itself. This implies (a) that only behavior encoded in the model can be tested and (b) that the different levels of abstraction must be bridged [5]: the input part of the test is concretized before it is fed into the SUT, and the output of the SUT is abstracted before it is compared to the expected output part of the test. For instance, the output of a model can be as abstract as "exception thrown" or "not thrown". Concretization and abstraction are performed by dedicated driver components. Coming up with the adequate abstraction continues to be an art.

Testing requires redundancy. Except for stress and performance testing, it is questionable to use one single model for the generation of both tests and code: the model would be tested against itself. In such a setting, only assumptions on the environment of the SUT and the code generator can be tested [5].

Promises and Benefits. Then dubbed "specification-based testing", the ideas of MBT have been around for about three decades. Traditionally, engineers form a vague understanding of the system by reading the specification. They build a mental model. Inventing tests on the grounds of these mental models is a creative process that is often implicit, barely reproducible, not motivated in its details and bound to the ingenuity of single engineers. Proponents of MBT claim that by making the mental model explicit, it is possible to generate sufficiently many tests of a sufficient quality at an acceptable cost in a structured manner. Even if it is rarely explicitly stated, the claim is that MBT yields better and cheaper tests than strategies that do not rely on explicit models (we do not discuss the relationship with reviews here; McConnell as well as Rombach and Endres have compiled studies that relate them to traditional testing [6, 7]).

The mere act of building behavior models in itself helps understand and clarify the requirements. This is because one is forced to thoroughly think about the system. In this vein, MBT can be seen as add-on to an activity the benefits of which usually go unchallenged, at least if cost is not an issue.

Assumptions. We will now make explicit three major assumptions that, usually concealed, go along with the first promise of the last paragraph.

The first assumption concerns models as abstractions. We have argued above that some loss of information is methodologically indispensable. The assumption consists of two parts. (1) Because models are simplifications, they are simpler to check than the SUT. (2) Sufficiently large parts of the omitted information can be re-inserted by the driver components that perform concretizations and abstractions: complexity can be distributed among model and driver.

The second assumption is concerned with the cost-effectiveness of the approach. It reads as follows. When compared to traditional forms of testing, the resulting high quality of a SUT that was tested on the grounds of a model justifies the cost of building the model and generating tests.

The third assumption concerns reuse. Recognizing the high cost of building models, it comes in two forms. One states that changes in the model as a consequence of changing requirements are (1) easy to validate, and, provided that MBT is a push-button technology, (2) re-generating tests is cheaper than changing existing test suites. The second form states that reusing models and test selection criteria in product lines automatically implies a reuse of tests.

Evidence. As far as we know, there is no published evidence that the promises of MBT are kept. Horstmann et al. have recently provided a compilation of case studies [8]. Roughly, all report on successful applications of MBT with automated test case generation in that "failures were found". With the exception of a recent study [2], comparative studies do not exist: MBT is not measured against random or manual testing (in fact, some studies do vaguely state that "model-based tests covered hand-written tests", without explicitly stating what exactly this means). Subject of the cited study [2] is the comparison of several test suites generated automatically/manually/randomly, all with and without a model. Comparison was done w.r.t. the number of detected failures, model coverage, and implementation coverage (condition/decision coverage). The main results of that study are (1) with a distinction between programming and specification errors, model-based tests detect significantly more specification errors than the other suites, (2) hand-crafted model-based tests detected roughly as many failures as a significantly larger number of automatically generated tests, and (3) inconclusive findings on the relationship between coverages and failure detection. The study is concerned with one single embedded system only, and it does not take into account the cost of building the model.

3 Conclusions

The ideas of MBT are appealing, in particular when testing is just one activity in an overall model-based development process. Numerous studies have shown that MBT does help with revealing errors, even in products that have been in the field for several years. We are convinced that MBT is a promising technology, but most studies leave the following questions open. (1) How many errors were detected during the modeling phase and before testing, i.e. during the careful review of the requirements or specification documents? (2) How many errors were detected by competitive technologies, in particular, testing without models and MBT with manual derivation of tests? (3) How do cost and benefits relate?

It is true that not all successful technology had had prior empirical evidence on its side (e.g. OO technology). To date, the question of whether or not structural test selection yield "better" tests than random tests remains undecided. We are convinced that an array of empirical studies on MBT for single systems will help find out where the technology is promising from a cost-benefit point of view, where it is not, and why. Further, comparative studies that take into account different technical approaches to MBT as well as different (kinds of) systems will help with identifying potential areas of successful application of MBT. This is even if the design of such studies is intricate, and even if, in sum, results of comparable work in the domain of error classifications are rather inconclusive [6]. A suite of benchmark problems with manually derived tests, known errors, and known cost of these errors could be a starting point.

References

1. Fagan, M.: Reviews and Inspections. In: Software Pioneers–Contributions to Software Engineering, Springer Verlag (2002) 562–573
2. Pretschner, A., Prenninger, W., Wagner, S., Kühnel, C., Baumgartner, M., Zölch, R., Sostawa, B., Stauner, T.: One evaluation of model-based testing and its automation. In: Proc. 27th Intl. Conf. on Software Engineering. (2005) To appear.
3. Zhu, H., Hall, P., May, J.: Software Unit Test Coverage and Adequacy. ACM Computing Surveys **29** (1997) 366–427

4. Musa, J.: Software Reliability Engineering. AuthorHouse, 2nd ed. (2004)
5. Pretschner, A., Philipps, J.: Methodological Issues in Model-Based Testing. Springer LNCS 3472. In: Model-Based Testing–a tutorial volume. (2005) 281–291
6. McConnell, S.: Code Complete. Microsoft Press (1993)
7. Endres, A., Rombach, D.: A Handbook of Software and Systems Engineering—Empirical Observations, Laws and Theories. Pearson Addison Wesley (2003)
8. Horstmann, M., Prenninger, W., El-Ramly, M.: Case Studies. Springer LNCS 3472. In: Model-Based Testing–a tutorial volume. (2005) 439–461