Dagstuhl Seminar
Perspectives of Model-Based Testing
September 5–10, 2004

Ed Brinksma – University of Twente
Wolfgang Grieskamp – Microsoft Research
Jan Tretmans – Radboud University Nijmegen

Software testing Software invades everywhere in our society and life, and we are increasingly dependent on it. This applies to all kinds of software: software in safety critical systems such as airplanes, in consumer products, in mobile phones, in telecom switches, in pace makers, in process control systems, in financial systems, in administration systems, etc. Consequently, the quality of software is an issue of increasing importance and growing concern.

Systematic testing is one of the most important and widely used techniques to check the quality of software. Testing, however, is often a manual and laborious process without effective automation, which makes it error-prone, time consuming, and very costly. Estimates are that testing consumes between 30-50% of the total software development costs. Moreover, the majority of the testing activities take place in the most critical phase of software development, viz. at the end of the project just before software delivery.

The tendency is that the effort spent on testing is still increasing due to the continuing quest for better software quality, and the ever growing size and complexity of systems. The situation is aggravated by the fact that the complexity of testing tends to grow faster than the complexity of the systems being tested, in the worst case even exponentially. Whereas development and construction methods for software allow the building of ever larger and more complex systems, there is a real danger that testing methods cannot keep pace with construction, so that these new systems cannot sufficiently fast and thoroughly be tested anymore. This may seriously hamper the development of future generations of software systems.

Model based testing One of the new technologies to meet the challenges imposed on software testing is *model-based testing*. In model-based testing a *model* of the desired behaviour of the *system under test* (SUT) is the starting point for testing. Model-based testing has recently gained attention with the popularization of modeling itself both in academia and in industry. The main virtue of model-based testing is that it allows test automation that goes well beyond the mere automatic execution of manually crafted test cases. It allows for the algorithmic generation of large amounts of test cases, including test oracles, completely automatically from the model of required behaviour. If this model is valid, i.e. expresses precisely what the system under test should do, all these tests are also provably

valid. Moreover, these models can, in principle, also be used for defining e.g. specification coverage metrics and test selection with mathematical rigour, so that quantifiable confidence is obtained, that a product faithfully conforms to its specification.

From an industrial perspective, model-based testing is a promising technique to improve the quality and effectiveness of testing, and to reduce its cost. The current state of practice is that test automation mainly concentrates on the automatic execution of tests. For this, a multitude of commercial test execution tools is available, but these tools do not address the problem of test generation. Model-based testing aims at automatically generating high-quality test suites from models, thus complementing automatic test execution.

From an academic perspective, model-based testing is a natural extension of formal methods and verification techniques, where many of the formal techniques can be reused. Formal verification and model-based testing serve complementary goals. Formal verification intends to show that a system has some desired properties by proving that a model of that system satisfies these properties. Thus, any verification is only as good as the validity of the model on which it is based. Model-based testing starts with a (verified) model, and then intends to show that the real, physical implementation of the system behaves in compliance with this model. Due to the inherent limitations of testing, such as the limited number of tests that can be performed, testing can never be complete: testing can only show the presence of errors, not their absence.

The interest in model-based testing from both industry and academia provides perspectives for academic-industrial cooperation in this area. This is also reflected in the relatively high industrial participation in the seminar, with researchers from Siemens, DaimlerChrysler, IBM, France Telecom, and Microsoft attending, and even co-organizing.

The Seminar The aim of the seminar *Perspectives of Model-Based Testing* was to bring together researchers and practitioners from industry and academia to discuss the state of the art in theory, methods, tools, applications, and industrialization of model-based testing, and to identify the important open issues and challenges.

In the past an analogous seminar had been organized: *Test Automation for Reactive Systems - Theory and Practice* (Ed Brinksma, Jan Peleska and Michael Siegel, Dagstuhl seminar 98361; Report 223; September 1998). Although at that time there were already quite a few research groups active in the area, it was only the beginning of developments in model-based testing. Significant progress has been made since then, and we had the impression that another Dagstuhl seminar on the topic was justified.

The presentations and discussions in the seminar addressed a broad spectrum of topics, which together gave a good overview of the current state of the art, the perspectives, and the open questions of model-based testing (MBT). Views from both academic and industrial perspectives were presented, different kinds of modelling and specification formalisms were used, and different test generation techniques, new ones as well as extensions of existing ones, were discussed. We present some of our observations.

- A couple of presentations came from industrial application domains: embedded automotive software, avionics, service platforms, telecom software, and general office software. It seems that at the moment the area of embedded and technical software is the most fertile domain for MBT, with Microsoft being the most notable exception. At Microsoft, applications of MBT go also beyond embedded and technical systems into the domain of general application programming interfaces and user interfaces.

The industrial presentations stressed that testing is an important aspect of software development, that automated testing is a necessity, and that MBT has a role to play. Moreover, they identified a couple of challenges and open issues:

  - It is important for MBT to deal the imperfect and incomplete real world, in which requirements are never complete, and specifications are always partial or loose.
  - The development of a software product is not a self-contained, one shot activity, but a software product evolves: it is iteratively developed in increments, it comes in different versions, configurations, and releases, and it is combined with other software products and components introducing all kind of compatibility problems. MBT should be able to cope with these issues.
  - An important question is how to obtain the formal models for MBT, and how to combine application domain knowledge and modelling knowledge. This issue is aggravated by the fact that there is no consensus yet about a suitable language for expressing these models.
  - An issue for many new techniques, and for MBT in particular, is the question of scalability.
  - To be successful, MBT must be integrated in the development process.
  - Not only detecting errors is important, but also locating and diagnosing errors. Combining MBT with with model-based diagnosis may be advantageous.

- Techniques to support MBT are drawn from many different areas like model checking, control and data flow analysis, static analysis, abstract interpretation, theorem proving, constraint solving, and run-time verification. These techniques are combined with traditional approaches to testing such as equivalence partitioning.

- The main emphasis during this seminar was on model-based testing of functionality. Other approaches to testing concerned user-profile based testing, where techniques like Markov-chain usage models and scenario-based statistical testing come into play.

- Whereas in the 1998-seminar a couple of basic, completely new test generation methods were presented, the current seminar had more emphasis on the investigation of extensions of these basic test generation methods, and on combinations of different methods and techniques. In particular, extensions for real-time testing, and combinations of state-based testing and data-oriented testing were elaborately discussed. Many of the presented techniques were in some sense "symbolic": symbolic test case generation, symbolic **ioco**, symbolic execution, symbolic transition systems, etc. Also extensions towards asynchronous testing, queue-based testing, and testing with action refinement were presented.

- Different kinds of languages and formalisms are used as the basis for MBT, such as FSM (Finite State Machines, or Mealy machines), different versions of labelled transition systems, (primitive recursive) functions, process algebra (e.g., CSP + CASL), ADT (Abstract Data Types), MSC (Message Sequence Charts), Spec#, etc. Most of them can be classified as formal, i.e., have a formal syntax and semantics, but also the less formal but industrially more often used notation UML was discussed as the basis for MBT. It was felt, however, that the lack of a precise semantics for UML, in particular of the dynamic behaviour part, hampers its use in model-based testing.

- Important, unsolved questions are how to select test cases, how to measure the quality of the selection, when to stop testing, how to quantify the remaining risk, and how to draw conclusions about the quality of a tested product. These problems are even

more prominent in MBT, since the quantity of generated test cases is almost unlimited. These issues, however, were not often addressed during seminar, which might be due to lack of progress on these important issues.

- To compare different MBT methods and tools it would be nice if there were a set of benchmarks, for example, sets of specifications, models, implementations, and mutants, to which different tools and methods could be applied.
- Controversy appeared on the question whether the average user should be aware of the complexity of using formal methods in general, and MBT in particular, or whether it should all be invisibly hidden in a tool, i.e., should MBT be push-button, or is this an infeasible dream?

Conclusion The presentations at the seminar gave a good insight into what has been achieved in the area of model-based testing, and, even more important, they gave clear indications of what has to be done before we can expect widespread industrial use of model-based testing.

Compared with the 1998-seminar, the area of model-based testing has certainly matured, with expanding interest also from industry. The feasibility of model based testing has been shown, more groups are involved, more theories are supported by tools, some of them close to industrially strength tools, and there are successful applications. Software testing has become a respectable research area.

The prospects for model-based testing to improve the quality and to reduce the cost of software testing are positive, but still more effort is needed, both in developing new theories and in making the existing methods and theories applicable, e.g., by providing better tool support.

To inspire a possible research agenda in model-based testing, we concluded the seminar with a discussion of a list of the top 10 challenging issues in model-based testing:

1. Measures for coverage and test quality: Model-based test generation algorithms can produce many test cases, but there is no method yet to compare the quality of two test suites. Quantification of test quality is desirable to compare test suites and to select the best one.
2. Test purposes and test scenario control: Often, it is necessary to guide or control the model-based generation of test cases so that the interesting, error-prone, or tricky parts of an SUT are really tested, i.e, the test purpose is reached. How to identify and specify this purpose, and how to control and guide the generation of tests is still unclear.
3. Merging different models: Many test generation methods work for particular aspects of behaviour, e.g., state-based models mainly test for control flow. For real systems many aspects must be tested at the same time: state, control flow, data flow, data transformation real-time, etc. This requires integration of the corresponding modelling formalisms, and of the test generation methods.
4. The role of quiescence, timed and untimed: An SUT should do what it is required to do. Doing nothing is a particular form of such a requirement. Doing nothing is technically expressed as being *quiescent*. Quiescence is treated differently in different test generation methods. Better understanding of quiescence is necessary, in particular, when real-time is involved.

5. Modelling method invocations: Many test generation methods have their origins in a message-oriented paradigm. Current component-based systems work differently: they are based on the object-oriented paradigm of method invocations. The modelling of method invocations for testing is not yet well-understood, in particular if parallelism and multi-threading is involved, so that several method invocations can exist concurrently.

6. Integration of techniques: The boundaries between such techniques as model-based testing, model checking, static analysis, abstract interpretation, theorem proving, constraint solving, run-time verification, etc. diminish. Integration of these techniques is necessary to be able to choose for every task the best combination of techniques.

7. Modelling test interfaces: For the execution of tests, the tester is connected to the SUT via some kind of test interface. The behaviour of this interface, e.g., an operating system *pipe* which behaves as a *FIFO queue*, must be taken into account when tests are generated. Research on different kinds of test interfaces and their influence on test generation and observation is desirable.

8. Tool architecture frameworks: Integration and interoperability of different (test) tools is desirable, e.g., interoperability of test generation tools and (on-the-fly) test execution tools.

9. Model based testing of non-functional properties: Most theory, methods, and tools for model-based testing have been devoted to testing of functionality. Model-based testing of other quality characteristics, such as security, reliability, performance, usability, etc., often referred to as non-functional properties, is an interesting field of research.

10. Promoting MBT in industry: To advance the industrial usage of MBT, it is necessary that the methods scale well to industrially sized problems, that they are sold with the right level of expectation, and that feedback from case studies is used in the next generation of MBT methods and tools.

The general opinion was that seminar was successful, and it was felt that, apart from the usual publication and presentation fora (e.g., FATES, ISSTA, MBT, TACAS, CAV, TestCom, ...) another Dagstuhl meeting on model-based testing should be organized in the future.

Ed Brinksma
Wolfgang Grieskamp
Jan Tretmans