A brief history of A-MOST Special Issue containing selected papers from A-MOST 2008

Lars Frantzen [a] Mercedes G. Merayo [b] Manuel Núñez [b]

[a] *Institute for Computing and Information Sciences - Informatics for Technical Applications, Radboud University Nijmegen, The Netherlands*

[b] *Dep. Sistemas Informáticos y Computación, Universidad Complutense de Madrid, Spain*

Abstract

This special issue contains the revised and extended versions of three papers presented in the 4th Workshop on Advances in Model-Based Testing (A-MOST 2008). In addition to an *executive summary* of these three papers, this preface briefly reviews the papers published in the four editions of the A-MOST workshop. We hope that the reader will find this special issue interesting and informative.

1 Introduction

The increasing use of software and the growing system complexity, in size, heterogeneity, autonomy, and physical distribution, makes software system testing a challenging task. Recent years have seen an increasing industrial and academic interest in the use of models for designing and testing software. Success has been reported in using a range of types of models, specification formats, notations and formal languages, such as UML, SDL, B and Z.

The use of models for designing and testing software is currently one of the most salient industrial trends with significant impact on the development and testing processes. Model-based tools and methods from object-oriented software engineering, formal methods, and other mathematical and engineering disciplines have been successfully applied and continue to converge into comprehensive approaches to software and system engineering.

The execution of software using test cases or sequences derived in a manual or automatic manner from models, often referred to as Model Based Testing (in short, MBT), is an encouraging scientific and industrial trend to cope with

growing software system complexity. Modeling requires a substantial investment, and practical and scalable MBT solutions can help leverage this investment. In this line, testing models may have been adapted from system design models or might have been devised specifically to support MBT. Naturally, the greatest benefits are often obtained when test generation is automated, but many practitioners report that the modeling process itself is of value, often highlighting requirements issues.

The use of industrial scale software demands the model-based construction of software and systems as compositions of independent and reusable actors. In this engineering paradigm, complex system functionality arises out of the composition of many component services. For these systems, MBT may significantly improve component acceptance and move component integration testing towards a canonical validation and certification of complete systems.

Automation of software development and software testing on the basis of executable models and simulation promises significant reductions in fault-removal cost and development time. As a consequence of automating MBT, changes in requirements analysis, development and testing processes are needed that demand combined efforts from research and industry towards a broadly accepted solution.

Since its creation, the *Workshop on Advances in Model-Based Testing*[1] (in short, A-MOST) has tried to be a referent in the area of MBT. The workshop has focused on three main areas: The models used in MBT; the processes, techniques, and tools that support MBT; and the evaluation of software using MBT and the evaluation of MBT itself. During 2009 the workshop celebrates its fifth edition, as a workshop collocated with the 2nd International Conference on Software Testing, Verification and Validation (ICST 2009). In fact, this is the first time that the workshop repeats collocation since its first four editions were collocated with four different conferences. After these five years, A-MOST has become a well established event in the area of model-based testing. In 2008 a new Steering Committee was elected by PC members of previous years. The current Steering Committee of A-MOST is conformed by:

- Lionel Briand, Simula Research Laboratory, Norway
- Robert M. Hierons, Brunel University, UK
- Manuel Núñez, Universidad Complutense de Madrid, Spain
- Alexander Pretschner, ETH Zurich, Switzerland

This special issue contains the extended and enhanced versions of three papers presented in the 2008 edition of A-MOST. The papers went through an additional reviewing process before they were finally accepted. Next we briefly

[1] The original name, used during the first two editions, was *Workshop on Advances in Model-Based Software Testing*.

review the context of these three papers.

2 Andrés, Llana and Rodríguez: Formally transforming user-model testing problems into implementer-model testing problems and viceversa

The first paper, by César Andrés, Luis Llana, and Ismael Rodríguez, explores two methods to assess the capability of a test suite to detect faults in a potentially wrong system, based on considering some probabilistic information. Testing is a major issue in any software development project because faults are indeed difficult to predict and find in real systems. Unfortunately, not finding any fault after applying some tests to the *Implementation Under Test* (IUT) does not imply, in general, that the IUT is correct, because the number of critical cases to be considered is typically infinite. For that reason, testers usually constrain the testing problem in several ways, or they assume some kind of *additional* information. One possibility it is to assume that the testers are provided with a probabilistic model of the *user* that will interact with the IUT. This model defines the probability that the external environment takes each available choice at each time. If the testers are provided with a probabilistic user model then they can assess the *suitability* of a finite set of tests: Those sets providing the highest *probabilistic coverage* of all possible interactions between the user and the IUT are preferable. Another possibility to constrain the problem is using a *fault model*, that is, a model denoting which IUT behaviors could be wrong indeed. If the testers are provided with such a model then tests only need to check those configurations marked as potentially faulty by the model. They can also consider a model of the *implementer* itself, instead of a model of the implementation. An implementer model is an abstract entity that tries to implement the structure defined by the specification and, at each step, it will be better those sets of tests such that their probability of detecting a fault, when applied to the IUT, is the highest. In this paper, the authors formally compare both probabilistic testing frameworks. After introducing both models and defining what an optimal test suite is in each case, they show that both models are strongly related indeed. In particular, it can be found the optimal test suite in one of these testing scenarios (i.e. either being provided with a user model, or having an implementer model) by solving the problem of finding the optimal test suite in the *other* problem, once a suitable transformation is performed in each case. That is, each of these problems can be *reduced* into the other problem. Beyond the relevance of this particular transformation, this result serves to illustrate a more general concept in testing: Testing problems can (non-trivially) be related to each other. Let us note that solving testing problems by using solutions designed for other testing problems may be relevant for two main reasons. On

the one hand, it implicitly provides a method to *export* our knowledge about how to solve some testing problems into another testing problems. On the other hand, such a reduction relation could provide us with a way to induce a *classification* of testing problems.

3 Madani and Parissis: Automatically testing interactive applications using extended task trees

The second paper, by Laya Madani and Ioannis Parissis, proposes a method for automatically generating test data from a task tree used to describe the interaction between a user and an interactive application. Interactive applications ensure the access to various commercial services (mobile phones, reservation systems or telecommunication services) and are increasingly involved in several critical domains such as flight or industrial process control. Therefore, their correctness becomes an important issue and their development requires thorough validation. Several automated methods have been proposed for verifying and validating interactive systems based on formal specifications. In most of these approaches, the interactive application is formally described as an abstract model and various properties which must hold are verified on this model by means of traditional verification techniques (e.g. model checking). MBT methods focusing on the formal specification of the user behavior have also been studied. A common concern of such approaches is that they require formal specifications that are not easy to provide for most interactive application designers. For this reason, test data generation methods based on usual models in the domain of interactive applications development should be studied. Task models are common notations used in the design of interactive software applications. Built by human factors specialists at early stages of the application design, they describe the interactions between the user and the application tasks. They contain valuable information about the expected user behavior as well as on the expected application reactions and thus, they can be used to support MBT. A well known notation for task modeling is ConcurTaskTrees (CTT). Tasks in CTT trees can be either elementary (user tasks or application tasks) actions or abstract tasks, composed of subtasks connected by means of temporal operators. In this paper, the use of CTT task trees for automatic test data generation purposes is studied. First, the CTT notation is extended to support operational profile specification. Such a feature makes possible to assign occurrence probabilities to the tasks. Thus, CTT task trees may describe several classes of execution scenarios corresponding to various user profiles (i.e. various occurrence probabilities). Next, the resulting extended task tree must be transformed into a model of the user behavior. This model extraction can be performed automatically, by applying formal composition rules, defined for every CTT operator. As a result, the user behavior

is encapsulated in a reactive model: a probabilistic finite input-output state machine. Generating test data requires simulating this probabilistic model. The simulation is made "on the fly": User actions are determined according to the application reactions with respect to the specified operational profiles. This simulation can be carried-out using Lutess, a testing environment developed for synchronous reactive software. The translation of the user interaction model into a Lutess test model is explained and experimental results are reported.

4 Fraser, Gargantini and Wotawa: On the order of test goals in specification-based testing

The third paper, by Gordon Fraser, Angelo Gargantini and Franz Wotawa, describes two sets of experiments performed in order to determine the impact of the order in which test goals are selected during test case generation, and explores the potential of simple heuristics to prioritize test goals. MBT techniques often select test cases according to test goals such as coverage criteria or mutation adequacy. Creating one test case per test goal can be a resource intensive task and result in large test suites. As one test case usually covers several test goals the testing costs can be reduced by either removing unnecessary test cases (test suite minimization) or by creating test cases only for unsatisfied test goals in the first place. The latter, however, causes the order in which test goals are selected to have an impact on the test suite size: In the theoretical best case a single test case might cover all test goals, while in the worst case one test case is created for each test goal. Finding a good order in which to select test goals has the potential to reduce the number of test cases and the time used to generate them. Unfortunately this is a difficult problem as finding an optimal order is not feasible in general because there are too many possible orderings. Comparison of the number of test cases, their total length, and the creation time in the initial set of experiments leads to the unexpected conclusion that the order matters less than expected. The second set of experiments aims to validate this result by using different model checkers and specifications and larger sample sets, but the conclusion is similar: Compared to the full test suite size (i.e. one test case per test goal) any order of the test goals will achieve a similar reduction. Compared to the minimal test suite size (i.e. removing any test case from the test suite will lead to unsatisfied test goals) there is some variation in the results which can justify prioritization of the test goals, for example when resources are limited and critical. Therefore, three simple heuristics are evaluated with respect to their effects on test suite size, length, and creation time, and the reduction compared to the mean value of random order is in the order of 5%. The heuristics are based on a temporal logic encoding of the test goals, so it is conceivable that a further improvement

is possible by including domain knowledge or information about the models.

5 A-MOST: The first four years

Next we briefly review the contributions of the participants in the four editions of A-MOST held up to now.

5.1 A-MOST 2005

The first edition of the A-MOST workshop was co-located with ICSE 2005 and held on 15-16 May 2005, in St. Louis, Missouri, USA. The chairmen [2] of this first event were Siddharta R. Dalal, Ashish Jain, and Jesse Poore. The proceedings of the workshop were published in ACM SIGSOFT Software Engineering Notes 30(64), 2005. Revised versions of four of these papers [36,10,4,40] appeared in a special issue: *Information & Software Technology* 48(10), 2006.

Renée C. Bryce and Charles J. Colbourn [9] presented analysis and algorithms to prioritize tests generated by the combinatorial design methods, where user-specified importance is treated as bias in the algorithms. Empirical evidence indicates that the bias results in potentially significant improvements in cumulative weight of test suites.

Markus Clermont and David Parnas [14] reported on selection of test cases based on specifications. More specifically, test case selection is a form of stress testing, which focuses on test cases drawn from interesting points in the input space, for example, points where properties of a function change. This work is in the stream of research from the Software Quality Research Lab of the University of Limerick and is based on the tabular representation of specifications.

Kirk Sayre [43] reported on a large undertaking to automate testing of a library of C++ class templates, the scientific code for computational materials research being developed and maintained at the Oak Ridge National Laboratory. The approach is based on Markov chains usage models and uses methods and tools developed in the Software Quality Research Lab at the University of Tennessee. While early in the overall program, the practicality and value of the approach have been demonstrated through numerous test cases and error discovery.

[2] We would like to thank them for allowing us to use their preface in order to review A-MOST 2005 papers.

David McGuiness and Liam Murphy [31] described a performance simulation model for servers built on enterpriser Java beans. The model has a set of user-modifiable parameters (e.g. memory size, CPU speed) and a set of model parameters based on the server and system (e.g. memory in use). The model computes statistics on performance and allows what-if analysis and prediction, based on parameter changes (e.g. increase in number of CPUs).

Erika Mir Olimpiew and Hassan Gomaa [33] described a method for producing test cases for an application derived from a software product line. The software product line spawns applications that have many features in common. Relationships between model elements of the product line and the application facilitate automatic selection and generation of tests.

Albert Schilling, Kelma Madeira, Paula Donegal, Kênia Sousa, Elizabeth Furtado, and Vasco Furtado [44] reported on test designs based on human-computer interaction. The work focuses on usability tests, such as the appropriateness of interface design to user preferences. Testing produces better user interfaces, and better user interfaces improve testability.

Christopher M. Lott, Ashish Jain, and Siddhartha R. Dalal [28] presented a tutorial through examples of the combinatorial approach to testing. Example system requirements and corresponding models for achieving pair wise coverage were presented. The terminology and notation are from the AETG.

Fabrice Bouquet, Eddie Jaffuel, Bruno Legeard, Fabien Peureux, and Mark Utting [8] reported on a requirements-traceability process successfully used in Smart Card applications. The process is based on annotating a formal model and then generating a traceability matrix calls for management of all the links among the requirements, the model, and the test cases. It is embedded in the LEIROS test generator. Effectiveness data on field application was presented.

Christopher Robinson-Mallett, Peter Liggesmeyer, Tilo Mücke, and Ursula Goltz [39] discussed shortest distinguishing sequences generated from a class of extended finite state machines. This algorithm is based on computation tree logic and can be integrated into any model checker that uses tree logic to generate test cases. Execution time complexity is not increased, but space requirements are.

Mikhail Auguston, James Bret Michael, and Man-tak Shing [3] described the use of attributed event grammars to model environments surrounding real-time systems. Environment models represented in this way can automatically generate a large number of pseudo random tests. A prototype of a test generator that takes a grammar and generates a test driver in the C language was developed.

Mark Sherriff, Nachiappan Nagappan, Laurie A. Williams, and Mladen A.

Vouk [45] discussed a research program centered on constructing and validating a set of process metrics useful for predicting external system defect density. Metric suites for Haskell were developed and field-tested. Field data collection and study methods were described for five independent variables, used in multiple regression analysis with random snapshots taken during the development cycle. Analysis showed that future defect densities could be predicted from historical data in the experimental situation.

Amit M. Paradkar [35] presented a two-part paper: A review of different classes of MBT techniques, followed by results from two case studies comparing techniques in terms of fault detection effectiveness. Case studies with methods that use mutation and explicitly generate state verification sequences had better fault detection than did those based on boundary values and predicate coverage criteria. Discussion of the role of test objectives and the problem of defining objectives followed.

Alan F. Karr and Adam A. Porter [22] recasted performance analysis as a model-based experimental design and execution problem. Commercial systems have hundreds of configuration parameters, combinations of which affect performance. The approach moves some parts of performance analysis from the laboratory to end-user systems.

Xia Cai and Michael R. Lyu [11] reported on hypotheses that predict the effect of code coverage testing on fault detection under different testing profiles. Using data from previous experiments, the work demonstrates that the effect of code coverage varies under different profiles.

Robert V. Binder and James E. Hanlon [6] presented a system for performing end-to-end testing of distributed mobile applications. The system uses model-based strategies to generate realistic test suites that represent external actors, system requirements, and the radio-frequency propagation environment of the system under test. Test input rates can be varied to represent aggregate workload. Post conditions can be specified for model parameters that can, in turn, be checked automatically with exception reporting.

Xing Li and Ramesh Nagarajan [27] reported on up-front modeling of image processing systems in an environment of short development cycles and high failure costs. By separating out concerns related to hardware-application interactions, the architecture of the model enables software to be tested even before the hardware becomes available.

Peter B. Lakey [26] reported on feasibility issues related to the use of MBT for a defense system. The large number of states in the model and the large number of test cases run in the demonstrations characterize the work. While demonstrating technical and economic feasibility, with significant benefits, the support tools need to be improved and extended before the application of MBT

can achieve widespread among defense contractors.

5.2 A-MOST 2006

The second edition of the A-MOST workshop was co-located with ISSRE 2006 and held on November 7, 2006, in Raleigh, North Carolina, USA. The chairmen of this edition were Mikhail Auguston and Bret Michael. The papers accepted in the workshop were published in ACM SIGSOFT Software Engineering Notes 31(6), 2006. Two of these papers [21,38] were included in the special issue: *Software Quality Journal* 16(2), 2008.

Myra B. Cohen, Joshua Snyder, and Gregg Rothermel [15] presented a case study that investigates the effects of changing configurations on two types of test suites. The results in this paper showed that test coverage and fault detection effectiveness do not vary much across configurations for entire test suites. However, for individual test cases and certain types of faults, configurations matter.

André L.L. de Figueiredo, Wilkerson de L. Andrade and Patrícia D. L. Machado [16] introduced a proposal to specify feature interaction requirements with use cases, by generating a behavioral model from such specification. They also provided a strategy for generating test cases from the behavioral model that aims to extract feature interaction scenarios in such a way that interactions can be tested.

Gordon Fraser and Franz Wotawa [18] introduced the notion of property relevance of test cases. Property relevant test cases can be used to determine property violations. They showed how to detect the properties relevant to a test case and new coverage criteria based on property relevance were introduced. Automated generation of test suites satisfying these criteria were also considered and feasibility was illustrated with an empirical evaluation.

Christopher Robinson-Mallett, Robert M. Hierons, and Peter Liggesmeyer [37] considered the problem of testing the communication between components of a timed distributed software system. They assume that communication is specified by using timed interface automata and use computation tree logic to define coverage criteria that refer to *send* and *receive* statements and communication paths. A major benefit of their approach is the generation of a potentially minimal set of test cases with the confidence that every interaction between components is executed during testing. They also provided a visual description of the state-based testing of distributed systems, which may be beneficial in other contexts such as education and program comprehension.

The third edition of the A-MOST workshop was co-located with ISSTA 2007 and held on July 9, 2007, in London, United Kingdom. The chairmen of this edition were Steve Counsell, Robert M. Hierons and Christopher Robinson-Mallett. The proceedings of the workshop were published by ACM Press.

Nicolas Kicillof, Wolfgang Grieskamp, Nikolai Tillmann, and Victor Braberman [23] devised a novel technique to automatically generate test cases for a software system, combining black-box MBT with white-box parameterized unit testing. The joint application of these techniques can produce test definitions that exercise all selected paths in the model, while also covering code branches specific to the implementation. These results cannot be obtained from any of the individual approaches alone, as the model cannot predict what values are significant to a particular implementation, while parameterized unit testing requires manually written test sequences and correctness validations.

Pierre-Alain Masson, Jacques Julliand, Jean-Chritophe Plessis, Eddie Jaffuel, and Georges Debois [30] proposed a solution to the problem of getting confident in the fact that an implementation correctly meets a security policy assigned to it. To do so, they compute tests that exercise security properties issued from the security policy. They use a functional model that formalizes the functional specification. But they also use a second model, in the shape of security properties, that formalize a part of the security policy. Tests are computed from the security properties, with the formal functional model as an oracle. The informal security requirements are formalized as regular expressions. Then, mutations are introduced in the regular expressions as to reflect the specific situations in which we intend to test the security properties. These mutated regular expression are unfolded into abstract test sequences. The authors present a set of four mutation rules that apply to a class of properties that we call sequencing properties, and experiment their method on a standard in the smart card domain named IAS, for Identification, Authentication and electronic Signature.

Sebastian Benz [5] presented an approach to use task models to describe the interaction between environment and system. When integrating different system components, the interaction between different features is often error prone. Typically errors occur on interruption, concurrency or disabling/ enabling between different features. Test case generation for integration testing struggles with two problems: The large state space and that these critical relationships are often not explicitly modeled. The proposal restricts the possible state space to a feasible size and enables the generation of task sequences, which cover the critical interaction scenarios. In order to overcome the different abstraction levels, tasks are mapped to component behavior models. Based on this

mapping, task sequences can be enriched with additional information from the component models and thereby executed to test the system under test.

Bogdan Korel, George Koutsogiannakis, and Luay H. Tahat [25] presented several model-based test prioritization heuristics. The major motivation to develop these heuristics was simplicity and effectiveness in early fault detection. During regression testing, a modified system needs to be retested using the existing test suite. Since test suites may be very large, developers are interested in detecting faults in the system as early as possible. Test prioritization orders test cases for execution to increase potentially the chances of early fault detection during retesting. Most of the existing test prioritization methods are based on the code of the system, but model-based test prioritization has been recently proposed. In model-based prioritization, information collected during execution of a model is used to prioritize tests for execution. The authors conducted a small experimental study in which they compared model-based test prioritization heuristics. The results shown that some simple heuristic methods can be as effective in early fault detection as more complex ones.

Qurat-ul-ann Farooq, Muhammad Zohaib Z. Iqbal, Zafar I Malik, and Aamer Nadeem [17] presented a UML based selective regression testing strategy that uses state machines and class diagrams for change identification. They identify the changes using the UML 2.1 semantics of state machines and class diagrams. The changes are classified as Class-driven (obtained from class diagrams) and State-driven (obtained from state machines). Class-driven changes are important as these changes are not reflected on the state machines and they might be helpful in identifying some fault-revealing test cases. With the help of the identified changes, the authors classify the test cases of the test suite as Obsolete, Reusable, and Retestable. The authors applied the approach on a case study to demonstrate its validity.

Yanping Chen, Robert L. Probert, and Hasan Ural [13] proposed a model-based regression test suite generation method based on Extended Finite State Machine (EFSM) dependence analysis. Twelve types of dependencies are identified related to three types of elementary modifications (EMs), that is, adding, deleting, and changing transitions in an EFSM model representing a system under test. These dependencies capture the effects of the model on the EMs, the effects of the EMs on the model, and the side-effects of the EMs. The proposed method constructs a regression test suite by covering all occurrences of these dependencies caused in a given EFSM model by a given set of EMs.

Gordon Fraser and Franz Wotawa [19] proposed the use of temporal logic rewriting techniques, which originate from runtime verification research, in order to overcome the problem that unnecessary, redundant calls to the model-checker cause bad performance.

Duminda Wijesekera, Paul Ammann, Lingya Sun, and Gordon Fraser [46] presented a formal relationship between counterexamples and test cases in the context of the computation tree logic, the logic of the popular model checker SMV. Given a test requirement as a computation tree logic formula, they defined what it means for a set of test cases to cover a counterexample associated with that requirement. This result cannot only be used in the generation of a test set that satisfies a given test coverage criterion, but also in the determination of whether an extant test set satisfies the criterion. The results can guide the production of counterexamples in model checkers explicitly intended to support testing.

Manoranjan Satpathy and S. Ramesh [42] adapted the abstraction refinement techniques of Clarke et al. and Gulavani et al. for systematic generation of model based test cases. Formal models are in general infinite state machines. The authors approximate such a formal model by a more abstract finite state machine. From the finite model, they obtain probable test cases through model checking, and then a guided symbolic execution is performed over the given formal model to check if this is a real test case. In case of failure, the finite abstract model is refined and the cycle is repeated. The two main results of this paper are that a better specification coverage is achieved, and a more accurate coverage is estimated. The method was explained by considering models in the specification language B.

F. Bouquet, C. Grandpierre, B. Legeard, F. Peureux, N. Vacelet, and M. Utting [7] presented a MBT approach that takes a UML behavioral view of the system under test and automatically generates test cases and executable test scripts according to model coverage criteria. This approach is embedded in the LEIRIOS Test Designer tool and is deployed in domains such as Enterprise IT and electronic transaction applications. This MBT approach makes it possible to automatically produce the traceability matrix from requirements to test cases as part of the test generation process. The work defines the subset of UML used for MBT and illustrates it using a small example.

Leila Naslavsky, Hadar Ziv, and Debra J. Richardson [32] proposed an approach that leverages model transformation traceability techniques to create fine-grained relationships among MBT artifacts. Testing against original expectations can be done with MBT that adopts high-level models as the basis for test generation. In addition to test generation, challenges to MBT include creation and maintenance of traceability information among test-related artifacts. Traceability is required to support activities such as result evaluation, regression testing and coverage analysis. Model-driven development and model transformation solutions address the traceability problem by creating relationships among transformed artifacts throughout the transformation process. In this work relationships are created during the test generation process. Their fine granularity enables the support for result evaluation, coverage analysis

and regression testing. Finally, Bernhard K. Aichernig, Martin Weiglhofer, Bernhard Peischl, and Franz Wotawa [1] evaluated two techniques for test purpose design: tests cases are designed on basis of structural coverage criterium and related to specific fault models. The authors present a heuristic algorithm for the extraction of tests cases for test purpose design and discuss the problem of overlapping of tests purposes. They illustrate improvements in terms of test execution times and in terms of number of test cases when minimizing this overlap. In addition, different strategies for the generation of fault-based test purposes are presented. For the evaluation they apply the presented techniques to a Session Initiation Protocol Registrar specification. All extracted test cases were executing against a commercial and open source implementation of such a Session Initiation Protocol Registrar.

5.4 A-MOST 2008

The fourth edition of the A-MOST workshop was co-located with ICST 2008 and held on April 9, 2008, in Lillehammer, Norway. The chairmen of this edition were Lars Frantzen, Mercedes G. Merayo and Manuel Núñez. The proceedings of the workshop were published by IEEE Computer Society in its digital library. In addition to the preliminary versions of the three papers contained in this special issue [2,29,20], the following papers appeared in the proceedings of the workshop.

Emine G. Paige and Jim Woodcock [34] aimed to improve the understanding of how to write useful validation scenarios for assertions written in the Object Constraint Language. Some approaches to MBT focus on test case generation from assertions (operation pre- and post- conditions) and invariants. In such a setting, assertions must be validated. Validation is often carried out via executing scenarios wherein system operations are applied, to detect unsatisfied invariants or failed preconditions. The authors report on their experiences during the creation and execution of 237 scenarios for validating assertions for the Mondex Smart Card application, and describe several observations that can help to improve the process of writing these scenarios. They also describe the important factors that must be considered in transforming these scenarios into abstract test cases.

Ana Cavalli, Eliane Martins and Anderson Morais [12] proposed a passive approach for robustness testing, in which the system under test is instrumented for fault injection during runtime, as well as for monitoring its behavior. Robustness testing has as main objective to determine how a system behaves in the presence of unexpected inputs or stressful environmental conditions. An approach commonly used for that purpose is fault injection, in which faults are deliberately injected into a system to observe its behavior. In the approach

proposed the readouts collected are analyzed to determine whether the observed behavior under faults is consistent with properties based on a finite state model of the system. The approach is illustrated using an implementation of the Wireless Application Protocol (WAP).

Pieter Koopman, Peter Achten, and Rinus Plasmeijer [24] showed how very desirable properties of specifications can be checked by systematic automated testing of the specifications themselves. The authors show how useful properties of specifications can be found by generalization of incorrect transitions encountered in simulation of the model. MBT of state based systems is known to be able to spot non-conformance issues. However, up to half of these issues appear to be errors in the model rather than in the system under test. Errors in the specification at least hamper the prompt delivery of the software, so it is worth while to invest in the quality of the specification. Worse, errors in the specification that are also present in the system under test cannot be detected by MBT. The authors show how useful properties of specifications can be found by generalization of incorrect transitions encountered in simulation of the model.

Finally, Valdivino Santiago, N. L. Vijaykumar, Danielle Guimares, and Ana Silvia Amaral [41] presented an environment, GTSC, which enables test sequences to be obtained from both Statechart-based and Finite State Machines-based behavioral models. The environment supports test case generation from some test methods for finite state machines, such as switch cover, DS and UIO methods, and also from some test criteria for Statecharts based on the SCCF family. Two case studies involving embedded software developed for two computers of scientific experiments of a satellite under development at National Institute for Space Research are presented in order to show the usefulness of the environment.

References

[1] B. K. Aichernig, M. Weiglhofer, B. Peischl, and F. Wotawa. Test purpose generation in an industrial application. In *3rd Workshop on Advances in Model-Based Testing, A-MOST'07*, pages 115–125. ACM Press, 2007. http://doi.acm.org/10.1145/1291535.1291547.

[2] C. Andrés, L. Llana, and I. Rodríguez. Formally comparing user and implementer model-based testing methods. In *4th Workshop on Advances in Model-Based Testing, A-MOST'08*, pages 1–10. IEEE Computer Society, 2008. http://doi.ieeecomputersociety.org/10.1109/ICSTW.2008.18.

[3] M. Auguston, J. B. Michael, and M. Shing. Environment behavior models for scenario generation and testing automation. In *1st Workshop on*

Advances in Model-Based Software Testing, A-MOST'05. ACM Press, 2005. http://doi.acm.org/10.1145/1082983.1083284.

[4] M. Auguston, J. B. Michael, and M. Shing. Environment behavior models for automation of testing and assessment of system safety. *Information & Software Technology*, 48(10):971–980, 2006.

[5] S. Benz. Combining test case generation for component and integration testing. In *3rd Workshop on Advances in Model-Based Testing, A-MOST'07*, pages 23–33. ACM Press, 2007. http://doi.acm.org/10.1145/1291535.1291538.

[6] R. V. Binder and J. E. Hanlon. The advanced mobile application testing environment. In *1st Workshop on Advances in Model-Based Software Testing, A-MOST'05*. ACM Press, 2005. http://doi.acm.org/10.1145/1082983.1083289.

[7] F. Bouquet, C. Grandpierre, B. Legeard, F. Peureux, N. Vacelet, and M. Utting. A subset of precise UML for model-based testing. In *3rd Workshop on Advances in Model-Based Testing, A-MOST'07*, pages 95–104. ACM Press, 2007. http://doi.acm.org/10.1145/1291535.1291545.

[8] F. Bouquet, E. Jaffuel, B. Legeard, F. Peureux, and M. Utting. Requirements traceability in automated test generation: application to smart card software validation. In *1st Workshop on Advances in Model-Based Software Testing, A-MOST'05*. ACM Press, 2005. http://doi.acm.org/10.1145/1082983.1083282.

[9] R. C. Bryce and C. J. Colbourn. Test prioritization for pairwise interaction coverage. In *1st Workshop on Advances in Model-Based Software Testing, A-MOST'05*. ACM Press, 2005. http://doi.acm.org/10.1145/1082983.1083275.

[10] R. C. Bryce and C. J. Colbourn. Prioritized interaction testing for pair-wise coverage with seeding and constraint. *Information & Software Technology*, 48(10):960–970, 2006.

[11] X. Cai and M. R. Lyu. The effect of code coverage on fault detection under different testing profiles. In *1st Workshop on Advances in Model-Based Software Testing, A-MOST'05*. ACM Press, 2005. http://doi.acm.org/10.1145/1082983.1083288.

[12] A. Cavalli, E. Martins, and A. Morais. Use of invariant properties to evaluate the results of fault-injection-based robustness testing of protocol implementations. In *4th Workshop on Advances in Model-Based Testing, A-MOST'08*, pages 21–30. IEEE Computer Society, 2008. http://doi.ieeecomputersociety.org/10.1109/ICSTW.2008.51.

[13] Y. Chen, R. L. Probert, and H. Ural. Model-based regression test suite generation using dependence analysis. In *3rd Workshop on Advances in Model-Based Testing, A-MOST'07*, pages 54–62. ACM Press, 2007. http://doi.acm.org/10.1145/1291535.1291541.

[14] M. Clermont and D. L. Parnas. Using information about functions in selecting test cases. In *1st Workshop on Advances in Model-Based Software Testing, A-MOST'05*. ACM Press, 2005. http://doi.acm.org/10.1145/1082983.1083276.

[15] M. B. Cohen, J. Snyder, and G. Rothermel. Testing across configurations: implications for combinatorial testing. In *2nd Workshop on Advances in Model-Based Software Testing, A-MOST'06*. ACM press, 2006. http://doi.acm.org/10.1145/1218776.1218785.

[16] A. L. L. de Figueiredo, W. de L. Andrade, and P. D. L. Machado. Generating interaction test cases for mobile phone systems from use case specifications. In *2nd Workshop on Advances in Model-Based Software Testing, A-MOST'06*. ACM press, 2006. http://doi.acm.org/10.1145/1218776.1218788.

[17] Q. Farooq, M. Z. Z. Iqbal, Z. I. Malik, and A. Nadeem. An approach for selective state machine based regression testing. In *3rd Workshop on Advances in Model-Based Testing, A-MOST'07*, pages 44–52. ACM Press, 2007. http://doi.acm.org/10.1145/1291535.1291540.

[18] G. Fraser and F. Wotawa. Property relevant software testing with model-checkers. In *2nd Workshop on Advances in Model-Based Software Testing, A-MOST'06*. ACM press, 2006. http://doi.acm.org/10.1145/1218776.1218787.

[19] G. Fraser and F. Wotawa. Using LTL rewriting to improve the performance of model-checker based test-case generation. In *3rd Workshop on Advances in Model-Based Testing, A-MOST'07*, pages 64–74. ACM Press, 2007. http://doi.acm.org/10.1145/1291535.1291542.

[20] G. Fraser and F. Wotawa. Ordering coverage goals in model checker based testing. In *4th Workshop on Advances in Model-Based Testing, A-MOST'08*, pages 31–40. IEEE Computer Society, 2008. http://doi.ieeecomputersociety.org/10.1109/ICSTW.2008.31.

[21] G. Fraser and F. Wotawa. Using model-checkers to generate and analyze property relevant test-cases. *Software Quality Journal*, 16(2):161–183, 2008.

[22] A. F. Karr and A. A. Porter. Distributed performance testing using statistical modeling. In *1st Workshop on Advances in Model-Based Software Testing, A-MOST'05*. ACM Press, 2005. http://doi.acm.org/10.1145/1082983.1083287.

[23] N. Kicillof, W. Grieskamp, N. Tillmann, and V. A. Braberman. Achieving both model and code coverage with automated gray-box testing. In *3rd Workshop on Advances in Model-Based Testing, A-MOST'07*, pages 1–11. ACM Press, 2007. http://doi.acm.org/10.1145/1291535.1291536.

[24] P. Koopman, P. Achten, and R. Plasmeijer. Testing and validating the quality of specifications. In *4th Workshop on Advances in Model-Based Testing, A-MOST'08*, pages 41–52. IEEE Computer Society, 2008. http://doi.ieeecomputersociety.org/10.1109/ICSTW.2008.43.

[25] B. Korel, G. Koutsogiannakis, and L. H. Tahat. Model-based test prioritization heuristic methods and their evaluation. In *3rd Workshop on Advances in Model-Based Testing, A-MOST'07*, pages 34–43. ACM Press, 2007. http://doi.acm.org/10.1145/1291535.1291539.

[26] P. B. Lakey. Model-based specification and testing applied to the ground-based midcourse defense (GMD) system: an industry report. In *1st Workshop on Advances in Model-Based Software Testing, A-MOST'05*. ACM Press, 2005. http://doi.acm.org/10.1145/1082983.1083291.

[27] X. Li and R. Nagarajan. Modeling for image processing system validation, verification and testing. In *1st Workshop on Advances in Model-Based Software Testing, A-MOST'05*. ACM Press, 2005. http://doi.acm.org/10.1145/1082983.1083290.

[28] C. M. Lott, Ashish Jain, and S. R. Dalal. Modeling requirements for combinatorial software testing. In *1st Workshop on Advances in Model-Based Software Testing, A-MOST'05*. ACM Press, 2005. http://doi.acm.org/10.1145/1082983.1083281.

[29] L. Madani and I. Parissis. Automated test of interactive applications using task trees. In *4th Workshop on Advances in Model-Based Testing, A-MOST'08*, pages 53–62. IEEE Computer Society, 2008. http://doi.ieeecomputersociety.org/10.1109/ICSTW.2008.8.

[30] P. Masson, J. Julliand, J. Plessis, E. Jaffuel, and G. Debois. Automatic generation of model based tests for a class of security properties. In *3rd Workshop on Advances in Model-Based Testing, A-MOST'07*, pages 12–22. ACM Press, 2007. http://doi.acm.org/10.1145/1291535.1291537.

[31] D. McGuinness and L. Murphy. A simulation model of a multi-server ejb system. In *1st Workshop on Advances in Model-Based Software Testing, A-MOST'05*. ACM Press, 2005. http://doi.acm.org/10.1145/1082983.1083278.

[32] L. Naslavsky, H. Ziv, and D. J. Richardson. Towards traceability of model-based testing artifacts. In *3rd Workshop on Advances in Model-Based Testing, A-MOST'07*, pages 105–114. ACM Press, 2007. http://doi.acm.org/10.1145/1291535.1291546.

[33] E. M. Olimpiew and H. Gomaa. Model-based testing for applications derived from software product lines. In *1st Workshop on Advances in Model-Based Software Testing, A-MOST'05*. ACM Press, 2005. http://doi.acm.org/10.1145/1082983.1083279.

[34] E. G. Paige and J. Woodcock. Observations for assertion-based scenarios in the context of model validation and extension to test case generation. In *4th Workshop on Advances in Model-Based Testing, A-MOST'08*, pages 11–20. IEEE Computer Society, 2008. http://doi.ieeecomputersociety.org/10.1109/ICSTW.2008.29.

[35] A. M. Paradkar. Case studies on fault detection effectiveness of model based test generation techniques. In *1st Workshop on Advances in Model-Based Software Testing, A-MOST'05*. ACM Press, 2005. http://doi.acm.org/10.1145/1082983.1083286.

[36] A. M. Paradkar. A quest for appropriate software fault models: Case studies on fault detection effectiveness of model-based test generation techniques. *Information & Software Technology*, 48(10):949–959, 2006.

[37] C. Robinson-Mallett, R. M. Hierons, and P. Liggesmeyer. Achieving communication coverage in testing. In *2nd Workshop on Advances in Model-Based Software Testing, A-MOST'06*. ACM press, 2006. http://doi.acm.org/10.1145/1218776.1218786.

[38] C. Robinson-Mallett, R. M. Hierons, J. H. Poore, and P. Liggesmeyer. Using communication coverage criteria and partial model generation to assist software integration testing. *Software Quality Journal*, 16(2):185–211, 2008.

[39] C. Robinson-Mallett, P. Liggesmeyer, T. Mücke, and U. Goltz. Generating optimal distinguishing sequences with a model checker. In *1st Workshop on Advances in Model-Based Software Testing, A-MOST'05*. ACM Press, 2005. http://doi.acm.org/10.1145/1082983.1083283.

[40] C. Robinson-Mallett, P. Liggesmeyer, T. Mücke, and U. Goltz. Extended state identification and verification using a model checker. *Information & Software Technology*, 48(10):981–992, 2006.

[41] V. Santiago, N. L. Vijaykumar, D. Guimares, and A. S. Amaral. An environment for automated test case generation from statechart-based and finite state machine-based behavioral models. In *4th Workshop on Advances in Model-Based Testing, A-MOST'08*, pages 63–72. IEEE Computer Society, 2008. http://doi.ieeecomputersociety.org/10.1109/ICSTW.2008.7.

[42] M. Satpathy and S. Ramesh. Test case generation from formal models through abstraction refinement and model checking. In *3rd Workshop on Advances in Model-Based Testing, A-MOST'07*, pages 85–94. ACM Press, 2007. http://doi.acm.org/10.1145/1291535.1291544.

[43] K. Sayre. Usage model-based automated testing of C++ templates. In *1st Workshop on Advances in Model-Based Software Testing, A-MOST'05*. ACM Press, 2005. http://doi.acm.org/10.1145/1082983.1083277.

[44] A. Schilling, K. Madeira, P. Donegan, K. Sousa, E. Furtado, and V. Furtado. An integrated method for designing user interfaces based on tests. In *1st Workshop on Advances in Model-Based Software Testing, A-MOST'05*. ACM Press, 2005. http://doi.acm.org/10.1145/1082983.1083280.

[45] M. Sherriff, N. Nagappan, L. A. Williams, and M. A. Vouk. Early estimation of defect density using an in-process Haskell metrics model. In *1st Workshop on Advances in Model-Based Software Testing, A-MOST'05*. ACM Press, 2005. http://doi.acm.org/10.1145/1082983.1083285.

[46] D. Wijesekera, P. Ammann, L. Sun, and G. Fraser. Relating counterexamples to test cases in CTL model checking specifications. In *3rd Workshop on Advances in Model-Based Testing, A-MOST'07*, pages 75–84. ACM Press, 2007. http://doi.acm.org/10.1145/1291535.1291543.