Robert M. Hierons Jonathan P. Bowen
Mark Harman (Eds.)

Formal Methods and Testing

An Outcome of the FORTEST Network
Revised Selected Papers

Springer

Volume Editors

Robert M. Hierons
Brunel University
School of Information Systems, Computing and Mathematics
Uxbridge, Middlesex UB8 3PH, UK
E-mail: rob.hierons@brunel.ac.uk

Jonathan P. Bowen
Mark Harman
King's College London, Department of Computer Science
Strand, London WC2R 2LS, UK
E-mail: jpbowen@gmail.com, mark.harman@kcl.ac.uk

Library of Congress Control Number: 2008923977

CR Subject Classification (1998): D.2, D.3, F.3, K.6

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN	0302-9743
ISBN-10	3-540-78916-2 Springer Berlin Heidelberg New York
ISBN-13	978-3-540-78916-1 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12249992 06/3180 5 4 3 2 1 0

Preface

With the growing significance of computer systems within industry and wider society, techniques that assist in the production of reliable software are becoming increasingly important. The complexity of many computer systems requires the application of a battery of such techniques. Two of the most promising approaches are formal methods and software testing. Traditionally, formal methods and software testing have been seen as rivals. Thus, they largely failed to inform one another and there was very little interaction between the two communities. In recent years, however, a new consensus has developed. Under this consensus, these approaches are seen as complementary. This has introduced the prospect of collaboration between individuals and groups in these fields and work such as that contained in this book.

This book, which came out of the Formal Methods and Testing (FORTEST) network [3], includes 12 peer-reviewed chapters on ways in which formal methods and software testing complement each other. FORTEST was formed as a network established under UK EPSRC funding that investigated the relationships between formal (and semi-formal) methods and software testing. In particular, it was concerned with ways in which these areas complement each other. This is an exciting area of research that has led to a number of significant results produced in both industry and academia. While the EPSRC funding for FORTEST was for a fixed period only, FORTEST is now a subject group of two BCS Special Interest Groups: Formal Aspects of Computing Science (BCS FACS) and Special Interest Group in Software Testing (BCS SIGIST).

FORTEST has considered a wide range of testing activities, including static testing techniques (such as model-checking and theorem proving) as well as dynamic testing. Its members have investigated a wide range of problems of interest to the testing and formal methods communities. Examples include: automating (black-box) test generation; producing tests likely (or guaranteed) to detect certain types of fault; using static analysis to support software testing; methods for reasoning about test effectiveness; producing tests that are likely to find faults in systems formally refined from specifications; and using tests to explore assumptions underlying proofs. While much of the work in this area has concerned the problem of generating tests from formal specifications and models, there is a much wider range of ways in which these fields might interact. In particular, there is the interesting question of the relationships between static testing and dynamic testing.

The relationship between formal methods and software testing has received significant attention since the late 1980s. Seminal work during this period includes the paper by Dick and Faivre on testing from a VDM specification [4] and work on testing from algebraic specifications (see, for example, [2, 6]), the latter introducing a framework. More recently, the term *model-based testing* has been introduced in order to describe an important aspect of this area (see, for example, [1, 5, 7, 11]). However, there is much earlier research on this topic. For example, in 1956 Moore used the term Gedanken-Experiments to describe testing and provided a conceptual framework in which to discuss testing from a finite state machine [10]. Later Hennie showed how test sequences can be produced from finite state machines [8].

In several important ways, software testing is unlike any other form of engineering testing process. Though it bears some similarities with other engineering testing activities, software testing differs because of the discrete behavior of the artifacts under test and the enormous variety of different scenarios for which testing must cater.

It is often said, quoting Dijkstra's famous aphorism, that testing can only reveal the presence of bugs, but not their absence. This has led some to conclude that testing is nothing more than an admission of defeat; why should energy be poured into such a doomed endeavor? This view misses two crucial points. Firstly, there is work that shows that formalized testing can indeed establish correctness, albeit with respect to a set of well-defined assumptions. Such testing can also be automated. Secondly, in many real-world situations, there is no viable alternative to testing. Therefore, there is a pressing need for research that addresses familiar engineering concerns; how can software testing become better, faster and cheaper? In this respect, testing is very much like other testing activities in any other engineering discipline.

Software testing reaches beyond software engineering and computer science, drawing in other disciplines that cover a very broad spectrum of activity. It involves such disparate elements as psychology, traditional engineering and even philosophy. This astonishing breadth has made the problems of software testing appealing to academics for several decades. However, this broadness has also, until recently, tended to dissuade researchers from more theoretical computer science research communities. Hitherto, there has been in these communities something of a perception of testing as an inherently informal, ad hoc activity that will not submit to any reasonable formal analysis.

This book aims to challenge this misconception, thereby stimulating wider interest in software testing problems within theoretical computer science research communities. The book consists of 12 chapters by leading researchers and groups in software testing and, in particular, in the problems of formalizing software testing techniques. This work shows how testing raises new questions for theoreticians: testing throws up new forms of interesting non-standard semantics, requires complexity analysis, builds on and uses formal specification notations, and can be combined with existing static analysis techniques. The book also shows how testing activities can be formalized so that it is possible to reason formally about the outcomes of a testing activity.

Testing is an activity designed to reveal bugs. As might be expected, there is plenty of challenging and exciting research in this volume on this aspect of software testing. However, the reader will also find examples of formalisms within which it is possible to prove correctness up to certain well-defined levels of abstraction, purely using testing. There is also much work on hybrid techniques that combine pure testing with other strands of computer science research. The crucial issue of automation is a recurrent theme. It is the key to meeting the engineering imperatives for testing in the real world. Such automation brings with it important formal proof obligations at the meta-level. That is, can we establish the correctness of the systems that will automate testing – *Quis custodiet ipsos custodes?*

Having finished the book, the reader will hopefully conclude that there is much about software testing that can be submitted to a formal analysis and that there is a great deal to be gained from such a formalization. It is also hoped that readers from the formal methods community will be motivated to consider software testing as a possible application for their work: there remain plenty of interesting, important unsolved problems in formal aspects of software testing.

Each of the 12 chapters in this book describes a way in which the study of formal methods and software testing can be combined in a manner that brings the benefits of formal methods (e.g., precision, clarity, provability) with the advantages of testing (e.g., scalability, generality, applicability).

Tretmans provides an overview of labelled transition systems (LTS), input output transition systems (IOTS) and the “ioco” input/output conformance relation. He also describes how test cases can be generated from an IOTS using ioco. Process algebras such as CCS, CSP, and LOTOS have been developed in order to specify or model systems in which there is a significant amount of concurrency. Since their semantics can be described in terms of LTSs there has been a significant amount of interest in testing using an LTS. Different notions of observations that can be made by the tester lead to a rich set of implementation relations and the test generation process depends upon the implementation relation used. The observation that there is an asymmetry between input and output led to the introduction of IOTs, which are essentially LTSs in which the labels/events are partitioned into inputs and outputs. The system under test (SUT) cannot block input and the environment cannot block output from the SUT: these observations place constraints which have an impact on testing and are encapsulated in the ioco implementation relation.

The chapter by Campbell et al. covers the Spec Explorer tool, developed and in use at Microsoft, which enables model-based testing of reactive systems in an object-oriented style. Testing is often undertaken in a haphazard manner and model-based testing allows a more rigorous and systematic approach to be taken. The development of the Spec Explorer tool has been demand-driven by users and it has been used for testing parts of operating systems, .NET components, etc. The chapter presents the concepts and foundational aspects of the tool in a formal manner. In conclusion, issues of importance for the development of the next generation of testing tools are considered.

Many systems have timing constraints and this has led to interest in specifying real-time systems and testing against such specifications. Hessel et al. describe approaches for testing against a timed input/output transition system (TIOTS). A TIOTS is essentially an LTS in which input and output are differentiated and time is included. The authors define an implementation relation rtioco that is similar to ioco, with real-time. They then show how tests can be developed offline (produced prior to test execution) or online (produced during test execution) and describe how these have been implemented in a tool based on the UPPAAL model-checker.

The chapters by Chen, Hierons and Ural and by Ammann, Offutt and Xu both concern testing based on finite state machines (FSMs). The chapter by Chen, Hierons and Ural concerns the problems of observability and controllability in testing state-based models using a distributed test environment. In such an environment, there are many coordination problems that present themselves. How do the separate testers know what input/output sequences are being observed by the other testers? This chapter shows how these problems can be overcome. The work is important because of the large number of systems that can be modelled using a state-based formulation and because of the increasing incidence of geographic separation, which increases the difficulty of testing such distributed state-based systems.

The chapter by Ammann, Offutt and Xu considers the problems of representation and achievement of coverage for systems modelled by state-based formulations and by graphical structures. The chapter considers the underlying graphical representation of the state-based models and addresses the question of what form of predicates should be extracted from these graphical representations. The paper draws together two strands of research on software testing: testing from FSM models and testing based on coverage of graphical representations of source code.

The chapter by Bogdanov provides a summary of the literature on testing from X-machines. X-machines are extended versions of FSMs. The extensions for X-machines are specifically tailored to testing and aim to make them inherently easy to test. Bogdanov provides a valuable introduction to this topic for the non-specialist. He is careful to avoid the detailed and somewhat dense formal notation that is often required in papers presenting specific contributions on X-machines, making this an excellent starting point for the interested outsider. This treatment makes the chapter ideal for the reader who wishes to discover more about this important and powerful approach to testing for state-based models.

The chapter by Gaudel and Le Gall presents an approach to the testing of data type implementations using an algebraic specification approach. Conformance may then be checked formally against the satisfaction of axioms. The approach can be generalized to specification methods that include data types in a variety of formalisms. The approach has been applied in both academic and industrial studies, typically for existing systems. The approach has been successful in identifying missing test cases that could have caused serious problems later if they had remained undiscovered until after deployment of the software.

The chapter by Vilkomir and Bowen describes testing criteria using the Z formal specification notation. It covers both the existing MC/DC (modified condition/decision coverage) criterion and also a new and stronger RC/DC (reinforced condition/decision coverage) criterion. The latter may be useful for critical software where an extra level of confidence is required. The formalization in Z revealed deficiencies and ambiguities in the informal description of MC/DC, allowing these to be given a specific meaning. A pleasing aspect of the specification of RC/DC is that the specification of MC/DC can be reused and augmented with an additional predicate to specify the stricter criterion. The chapter includes an example to demonstrate use of the criteria and an illustration of why RC/DC could be beneficial in a system where the highest level of assurance is required.

There are many different test criteria and test generation techniques and this naturally leads to the question of which is best. While "best" depends on a number of factors and can refer to issues of cost as well as effectiveness, the focus to date has been on effectiveness. Weyuker describes the attempts that have been made to theoretically compare test criteria and points to the deficiencies in these approaches. One problem identified with all of these approaches is that they refer to idealized versions of test criteria and techniques rather than how testers actually use them. Weyuker argues that the only way of overcoming this is through empirical evaluation using case studies and experiments, but also points to the problems of carrying out experiments in software engineering (e.g., what are typical programs, specifications, faults, etc?). She concludes by suggesting that real progress is only likely to be made through a combination of empirical evaluation through case studies and advances in the theory.

The chapter by Schieferdecker et al. covers the TTCN-3 testing technology, which is used widely in the telecommunications sector. TTCN-3 is a new textual and graphical notation with a precise syntax, based on the earlier TTCN (tree and tabular combined notation). The language was developed by industry and academia to allow black-box and gray-box testing. It enables both test specification and execution. TTCN-3 has an intuitive operational semantics and is typically used for testing conformance and interoperability of protocols. However, it is also applicable to functionality testing of systems based on software. It is often used in safety-critical domains. The chapter gives an overview of the TTCN-3 language and some examples of use.

The chapter by Harman et al. concerns the transformation of programs to improve testability. Perhaps counter-intuitively, the approach does not maintain the original traditional semantics of the program under test; rather, it suggests the existence of interesting non-standard semantics. The transformations seek to improve the performance of testing for a particular test data generation technique, using a combination of standard and novel transformation rules. The chapter provides both an introduction to the area and also a set of open problems still to be solved. Thus it is suitable for both those requiring an introduction to the approach and also those wishing to develop the area further as part of their own research.

The chapter by Littlewood, Popov, Strigini and Shryane is concerned with techniques for locating faults in software. There are many such fault-finding techniques in the literature and a test manager would be forgiven for deciding that they should simply use several of these to achieve a “belt and braces approach” to their test effort. This chapter reveals that there are subtle dependencies between the different fault-finding techniques that mean one cannot necessarily make straightforward judgments as to the overall effectiveness of the combined techniques. To avoid this problem, the authors show how the concept of diversity of techniques can be defined formally and used to ensure that the power of the combined techniques more closely approaches the sum of the power of the separate parts, as the test manager might have hoped.

We would like to thank the authors for their contributions, without which this book would not have been possible. Thanks are also due to the reviewers for their detailed, constructive reviews of each chapter and to the staff of Springer for their assistance and guidance in seeing this volume through to publication. This work grew out of the editors’ involvement in and management of the United Kingdom EPSRC-funded FORTEST project. FORTEST, the FORmal Methods and TESTing Network, drew together the testing and formal methods communities in a network of research meetings, workshops and other events from 2001 to 2005. For further information, see *www.fmnet.info/fortest*.

We hope that you will find this volume stimulating reading and that the contributions herein might raise awareness in the testing community of the potential applications for formal methods and also that it may raise interest in testing as an application area for the formal methods community. There is much that these two, largely disjoint, research communities have to gain. The FORTEST network and other related activities have gone some way to bringing the two together. This book is one fruit of that union. We sincerely hope that there will be many more to come.

February 2008

Robert M. Hierons
Jonathan P. Bowen
Mark Harman

Acknowledgements. Selected authors of papers reviewed the chapters for this collected volume. These included: Paul Ammann, David Binkley, Jessica Chen, Marie-Claude Gaudel, Anders Hessel, Pascale La Gall, Jens Grabowski, Wolfgang Grieskamp, Bogdan Korel, Kim Larsen, Bev Littlewood, Brian Nielsen, Jeff Offutt, Paul Pettersson, Nikolai Tillmann, Jan Tretman, Hasan Ural, Margus Veanes, Sergiv Vilkomir. The FORTEST Network on formal methods and testing [3], of which all the editors were members, was funded by the UK Engineering and Physical Sciences Research Council (EPSRC) under grant number GR/R43150 from 2001 to 2005. Chapter 12 originally appeared in the *IEEE Transactions on Software Engineering* journal [9] and is reproduced here with permission.

References

1. Barnett, M., Grieskamp, W., Nachmanson, L., Schulte, W., Tillmann, N., Veanes, M.: Towards a tool environment for model-based testing with AsmL. In: Petrenko, A., Ulrich, A. (eds.) FATES 2003. LNCS, vol. 2931, pp. 252–266. Springer, Heidelberg (2004)
2. Bouge, L., Choquet, N., Fibourg, L., Gaudel, M.-C.: Test sets generation from algebraic specifications using logic programming. Journal of Systems and Software 6(4), 343–360 (1986)
3. Bowen, J.P., Bogdanov, K., Clark, J., Harman, M., Hierons, R., Krause, P.: FORTEST: Formal methods and testing. In: Proc. 26th Annual International Computer Software and Applications Conference (COMPSAC 2002), Oxford, UK, August 26–29, 2002, pp. 91–101. IEEE Computer Society Press, Los Alamitos (2002)
4. Dick, J., Faivre, A.: Automating the generation and sequencing of test cases from model-based specifications. In: Larsen, P.G., Woodcock, J.C.P. (eds.) FME 1993. LNCS, vol. 670, pp. 268–284. Springer, Heidelberg (1993)
5. Farchi, E., Hartman, A., Pinter, S.: Using a model-based test generator to test for standard conformance. IBM Systems Journal 41(1), 89–110 (2002)
6. Gaudel, M.-C.: Testing can be formal too. LNCS, vol. 915, pp. 82–96. Springer, Heidelberg (1995)
7. Grieskamp, W.: Multi-paradigmatic model-based testing. In: Havelund, K., Núñez, M., Roşu, G., Wolff, B. (eds.) FATES 2006 and RV 2006. LNCS, vol. 4262, pp. 1–19. Springer, Heidelberg (2006)
8. Hennie, F.C.: Fault-detecting experiments for sequential circuits. In: Proceedings of Fifth Annual Symposium on Switching Circuit Theory and Logical Design, Princeton, New Jersey, pp. 95–110 (November 1964)
9. Littlewood, B., Popov, P.T., Strigini, L., Shryane, N.: Modeling the effects of combining diverse software fault detection technique. IEEE Transactions on Software Engineering 26(12), 1157–1167 (2000)
10. Moore, E.P.: Gedanken-experiments. In: Shannon, C., McCarthy, J. (eds.) Automata Studies, Princeton University Press, Princeton (1956)
11. Utting, M., Legeard, B.: Practical Model-Based Testing: A Tools Approach. Morgan Kaufmann Publishers (2007)

Table of Contents