(12) Overview

FORmal methods and TESTing (FORTEST)

Dr R. M. Hierons, Proposal Coordinator

1 Project Overview

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. FORTEST is a cross-community network that will bring together expertise from each of these two fields.

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 [14]. This opens up the prospect of collaboration between individuals and groups in these fields.

While there has already been some work on generating tests from formal specifications and models, FORTEST will consider a much wider range of ways in which these fields might interact. In particular, it will consider relationships between *static testing* (verification that does not involve the execution of the implementation) and *dynamic testing* (executing the implementation).

FORTEST will build a new community that will explore ways in which formal methods and software testing complement. This will allow these fields to inform one another in a systematic and effective manner and thus facilitate the development of new approaches and techniques that assist the production of high quality software. The significance of this topic and the lack of any large UK groups, working on links between testing and formal methods, make it vital that such a network is established in the near future.

This proposal is timely because of the recent increased national and international interest in this subject which makes the formation of a community both feasible and desirable. It is anticipated that the existence of this network will lead to further collaboration and thus to a significant increase in the quality and quantity of research produced in this area.

The main aims of FORTEST are

- to bring together academics and industrialists interested in formal methods and software testing;
- to stimulate collaboration between individuals and groups in these fields;
- to disseminate problems and results to researchers and practitioners in these two fields and to the wider Software Engineering community.

The dissemination of this information will lead to a greater awareness of the links between software testing and formal methods. It will also widen the use of methodologies that assist the development of reliable software.

The network will focus on the following problems.

- How can the relationships between formal methods and software testing be utilised?
- How can the software development process be adapted in ways that simplify the utilisation of these relationships?
- How can techniques, developed to utilise the relationships between formal methods and testing, be automated?

1.1 Background

The use of a formal specification or model eliminates ambiguity and thus reduces the chance of errors being introduced during software development. Where a formal specification exists, both the source code and the specification may be seen as formal objects that can be analysed and manipulated. The use of a formal specification thus introduces the possibility of the formal and, potentially, automatic analysis of the relationship between the specification and the source code. This is often assumed to take the form of a proof, but such a

proof cannot guarantee operational correctness. For this reason, even where such a proof exists, it is important to apply dynamic testing [8].

Testing may be seen as any process that provides information that might either detect faults or provide confidence in the implementation under test (IUT). Thus, as noted earlier, there are both static and dynamic test techniques. Each form of testing provides some information about the IUT.

Static and dynamic testing have very different characteristics and so provide very different types of information. Static testing relies upon some underlying model that describes the link between the source code and the behaviour exhibited by the IUT. This model might, for example, be the semantics of the programming language and in this case any analysis relies upon the correctness of the compiler and hardware. Static testing may provide general information about a model of the system, but cannot be applied directly to the IUT. Dynamic testing, in contrast, provides specific information about the IUT. One challenge is thus to combine these two forms of testing in order to provide general information about the IUT.

1.1.1 Formal Methods and Dynamic Testing Complement

Software testing is an important and, traditionally, extremely expensive part of the software development process. Studies suggest that testing often forms in the order of fifty percent of the total development cost [3]. Where formal specifications and models exist, these may be used as the basis for automating parts of the testing process [1, 2, 19, 6, 21, 20, 10, 11, 5, 16, 22, 13]. This may lead to more efficient and effective testing. It may thus transpire that the automation of parts of the software testing process is one of the most significant benefits of using a formal specification language. The links between testing and formal methods do, however, go well beyond generating tests from a formal specification.

The presence of a formal specification or model makes it possible for the tester to be clearer about what it means for a system to pass a test. This may be achieved through the use of test hypotheses [9] or design for test conditions [17, 18, 15]. Similar ideas may be found in the generation of checking experiments from finite state machines [4, 23, 12]. Using these approaches it is possible to generate tests that determine correctness under certain well understood conditions circumventing Dijkstra's famous aphorism that testing can show the presence of bugs, but never their absence [7]. Program analysis might be used in order to either prove that these conditions hold or to provide confidence in these conditions holding.

Information gathered by dynamic testing may assist when using a formal specification. Testing may be used in order to provide initial confidence in a system before effort is expended in attempting to prove correctness. Where it is not cost effective to produce a proof of conformance, the developers may gain confidence in the implementation through systematic testing. This might be complemented by proofs that certain critical properties hold. A proof of correctness might also use information derived during testing. Finally, a proof of correctness relies upon a model of the underlying system and dynamic testing might be used to check this model. An interesting challenge is to generate tests that are likely to be effective in detecting errors in the assumptions inherent in a proof of correctness.

2 The goals of FORTEST

FORTEST is a three year project that will develop a new community in order to investigate ways in which the relationships between formal methods and software testing may be exploited. It will raise the awareness of the links between these fields and disseminate new methods and techniques developed by the community. The main goals are thus

1. to develop a community;
2. to disseminate current results and problems and new results produced by this community.

The deliverables of FORTEST include:

1. a workshop to be held approximately six months after the start of the project in order to establish an initial community and to formalise the set of problems;
2. a workshop approximately eighteen months after the start of the project in order to widen the range of problems considered and approaches used;

3. a workshop in the last six months of the project to disseminate the results;

4. research meetings, that are internal to the network, to be held quarterly (except where they would coincide with a workshop);

5. an internet structure that includes a mailing list and a web site containing information about the project.

References

[1] A. V. Aho, A. T. Dahbura, D. Lee, and M. U. Uyar. An optimization technique for protocol conformance test generation based on UIO sequences and Rural Chinese Postman Tours. In *Protocol Specification, Testing, and Verification VIII*, pages 75 86, Atlantic City, 1988. Elsevier (North-Holland).

[2] N. Amla and P.Ammann. Using Z specifications in category partition testing. In *COMPASS '92, Seventh Annual Conference on Computer Assurance*, pages 15 18, Gaithersburg, MD, USA, 1992.

[3] B. W. Boehm. *Software Engineering Economics.* Prentice-Hall, 1981.

[4] T. S. Chow. Testing software design modelled by finite state machines. *IEEE Transactions on Software Engineering*, 4:178 187, 1978.

[5] J. Derrick and E. Boiten. Testing refinements of state-based formal specifications. *Journal of Software Testing, Verification, and Reliability*, 9:27 50, 1999.

[6] J. Dick and A. Faivre. Automating the generation and sequencing of test cases from model-based specifications. In *FME '93, First International Symposium on Formal Methods in Europe*, pages 268 284, Odense, Denmark, 19-23 April 1993. Springer-Verlag, Lecture Notes in Computer Science 670.

[7] E. W. Dijkstra. Notes of structured programming. In O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, editors, *Structured Programming.* Academic Press, 1972.

[8] J. H. Fetzer. Program verification: The very idea. *Communications of The ACM*, 31:1048–1063, 1988.

[9] M. C. Gaudel. Testing can be formal too. In *TAPSOFT'95*, pages 82 96. Springer-Verlag, March 1995.

[10] R. M. Hierons. Testing from a finite state machine: Extending invertibility to sequences. *The Computer Journal*, 40:220 230, 1997.

[11] R. M. Hierons. Testing from a Z specification. *Journal of Software Testing, Verification and Reliability*, 7:19 33, 1997.

[12] R. M. Hierons. Adaptive testing of a deterministic implementation against a nondetermistic finite state machine. *The Computer Journal*, 41:349–355, 1998.

[13] R. M. Hierons, S. Sadeghipour, and H. Singh. Testing a system specified using Statecharts and Z. *Information and Software Technology*, 43:137–149, 2001.

[14] C. A. R. Hoare. How did software get so reliable without proof? In *Proceedings of Formal Methods Europe, 96 (Lecture Notes in Computer Science 1051)*, pages 1 17. Springer-Verlag, 1996.

[15] M. Holcombe and F. Ipate. *Correct Systems: Building a Business Process Solution.* Springer-Verlag, 1998.

[16] Hyoung Seok Hong, Young Gon Kim, Sung Deok Cha, Doo Hwan Bae, and Hasan Ural. A test sequence selection method for statecharts. *Journal of Software Testing, Verification and Reliability*, 10:203–227, 2000.

[17] F. Ipate and M. Holcombe. An integration testing method that is proved to find all faults. *International Journal of Computer Mathematics*, 63:159 178, 1997.

[18] F. Ipate and M. Holcombe. A method for refining and testing generalised machine specifications. *International Journal of Computer Mathematics*, 68:197 219, 1998.

[19] G. Laycock. Formal specification and testing: A case study. *Journal of Software Testing, Verification and Reliability*, 2:7–23, 1992.

[20] H. Singh, M. Conrad, and S. Sadeghipour. Test case design based on Z and the classification-tree method. In *First IEEE Conference on Formal Engineering Methods*, pages 81 90, Hiroshima, Japan, November 1997. IEEE Computer Society.

[21] P. Stocks and D. Carrington. A Framework for Specification-Based Testing. *IEEE Transactions on Software Engineering*, 2:777–793, 1996.

[22] H. Ural, K. Saleh, and A. Williams. Test generation based on control and data dependencies within system specifications in SDL. *Computer Communications*, 23:609–627, 2000.

[23] H. Ural, X. Wu, and F. Zhang. On minimizing the lengths of checking sequences. *IEEE Transactions on Computers*, 46:93 99, 1997.