Interpreting a successful testing process: risk and actual coverage ⋆

Mariëlle Stoelinga and Mark Timmer

University of Twente, Department of Computer Science, The Netherlands
Formal Methods & Tools
{marielle, timmer}@cs.utwente.nl

Abstract. Testing is inherently incomplete; no test suite will ever be able to test all possible usage scenarios of a system. It is therefore vital to assess the implication of a system passing a test suite. The work discussed in this presentation quantifies that implication by means of two distinct, but related, measures: the *risk* quantifies the confidence in a system after it passes a test suite, i.e. the number of faults still expected to be present (weighted by their severity); the *actual coverage* quantifies the extent to which faults have been shown absent, i.e. the fraction of possible faults that has been covered. We apply a probabilistic approach, taking into account uncertainties in system behaviour. We provide evaluation algorithms that calculate the metrics for a given test suite, as well as optimisation algorithms that yield the best test suite for a given optimisation criterion.

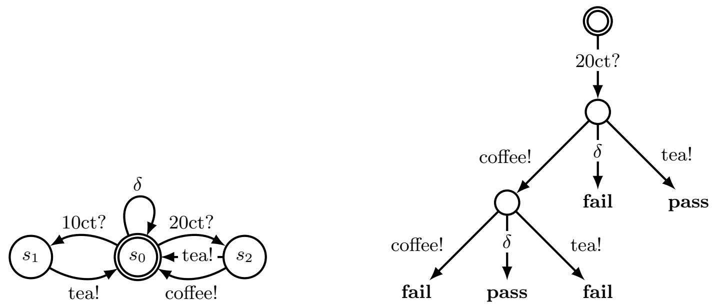
1 Problem description

Software becomes more and more complex, making thorough testing an indispensable part of the development process. The U.S. National Institute of Standards and Technology has assessed that software faults cost the American economy almost sixty billion dollars annually. More than a third of these costs could be eliminated if testing occurred earlier in the development process [14].

An important fact about testing is that it is inherently incomplete, since testing everything would require infinitely many input scenarios. On the other hand, passing a well-designed test suite does increase the confidence in the correctness of the tested product. Therefore, it is important to assess the quality of a test suite. Two fundamental concepts have been put forward to evaluate test suite quality: (1) *coverage metrics* determine which portion of the requirements and/or implementation-under-test has been exercised by the test suite; (2) *risk-based metrics* assess the risk of putting the tested product into operation.

Although existing coverage measures, such as code coverage in black-box testing [3, 12] and state and/or transition coverage in white-box testing [8, 13, 17], give an indication of the quality of a test suite, it is not necessarily true that higher coverage implies that more, or more severe, faults are detected. This is

⋆ To be presented by Mark Timmer.



(a) An IOLTS $\mathcal{A}$

(b) A test case for $\mathcal{A}$

Fig. 1. An IOLTS and a test case

because these metrics do not take into account where in the system faults are most likely to occur. Risk-based testing methods do aim at reducing the expected number of faults, or their severity. However, these are often informal [15], based on heuristics [2], or indicate which components should be tested best [1], but rarely quantify the risk after a successful testing process in a precise way.

2 Approach

We present a framework in which test coverage and risk can be defined, computed and optimised. Key properties are a rigorous mathematical treatment based on solid probabilistic models, and the result that higher coverage (resp. lower risk) implies a lower expected number of faults.

Starting point in our theory is a *weighted fault specification (WFS)*, which consists of a specification describing the desired system behaviour as an input-output labelled transition systems (IOLTSs), a weight function that describes the severity of the faults, an error function describing the probability that a certain error has been made, and a failure function describing the probability that incorrectly implemented behaviour yields a failure. This failure function is necessary since observing a correct response once does not yet imply correctness, as results may be nondeterministic. An example of an IOLTS with one of its possible test cases is shown in Figure 1.

From the WFS we derive its underlying probability model, i.e. a random variable that exactly describes the distribution of erroneous implementations. This allows us to define risk and actual coverage in an easy and precise way. Given a WFS, we define the *risk* of a test suite as the expected fault weight that remains after this test suite passes. We will show how to construct a test suite of a certain size with minimal expected risk.

We also introduce *actual coverage* for a test suite, which quantifies the risk reduction after passing the test suite. Whereas the risk is based on faults contained within the entire system, actual coverage only relates to the part of a system we tested. This coincides with the traditional interpretation of coverage. Optimisation with respect to actual coverage can be done in exactly the same way as optimisation with respect to risk.

Our methods refine [5], which introduces a concept we would call *potential coverage*, since it considers which faults *can* be detected during testing. Our measures, on the other hand, take into account the faults that *are* actually covered during a test execution.

While error probabilities and failure probabilities are important ingredients in our framework, techniques for obtaining these probabilities fall outside the scope of this presentation. However, there is extensive literature on factors that determine them. For instance, as described in [6], estimating the error probabilities can be based on the software change history. They can also be based on McCabe's cyclomatic complexity number [10], Halstead's set of Software Science metrics [7], and requirements volatility [9]. The failure probabilities can be obtained by applying one of the many analysis techniques described in [11] and [18]. In practice it might still be difficult to estimate all the probabilities that are needed, asking for simplifying approximations. Our theory could then serve as a baseline for sensitivity analysis [16], making it possible to assess the impact of these simplifications.

3 Conclusions and opportunities

While testing is an important part of today's software development process, little research has been devoted to the interpretation of a successful testing process. The work discussed in this presentation describes how to specify systems and their probabilistic behaviour. Based on such specifications, two measures can be calculated. Risk denotes the confidence in the system after testing is successful, whereas actual coverage denotes how much was tested.

Our work gives rise to several directions for future research. First, it is crucial to validate our framework by means of creating tool support and applying this in a series of case studies. Second, it might be useful to include fault dependencies in our models. Third, it is interesting to investigate the possibilities of on-the-fly test derivations (as e.g. performed by the tool TorX [4]) based on risk or actual coverage. This could yield a tool that, during testing, calculates probabilities and decides which branches to take for an optimal result. Finally, our framework might be used to investigate the sensitivity of the probabilities that are used. That way, simplifying approximations for risk and actual coverage could be validated.

References

[1] S. Amland. Risk-based testing: risk analysis fundamentals and metrics for software testing including a financial application case study. *Journal of Systems and Software, Volume 53, Issue 3*, pages 287–295, 2000.

[2] J. Bach. Heuristic risk-based testing. *Software Testing and Quality Engineering Magazine*, November 1999.

[3] T. Ball. A Theory of Predicate-Complete Test Coverage and Generation. In *Proc. of the 3rd Int. Symp. on Formal Methods for Components and Objects (FMCO '04)*, volume 3657 of *LNCS*, pages 1–22, 2004.

[4] A. Belinfante, J. Feenstra, R.G. de Vries, J. Tretmans, N. Goga, L.M.G. Feijs, S. Mauw, and L. Heerink. Formal test automation: A simple experiment. In *Proc. of the 12th Int. Workshop on Testing Communicating Systems (IWTCS '99)*, volume 147 of *IFIP Conference Proceedings*, pages 179–196. Kluwer, 1999.

[5] L. Brandán Briones, E. Brinksma, and M.I.A. Stoelinga. A semantic framework for test coverage. In *Proc. of the 4th Int. Symp. on Automated Technology for Verification and Analysis*, volume 4218 of *LNCS*, pages 399–414. Springer, 2006.

[6] T.L. Graves, A.F. Karr, J.S. Marron, and H. Siy. Predicting fault incidence using software change history. *IEEE Trans. on Softw. Eng.*, 26(7):653–661, 2000.

[7] M.H. Halstead. *Elements of Software Science*. Elsevier Science, 1977.

[8] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines - a survey. In *Proc. of the IEEE*, volume 84, pages 1090–1126, 1996.

[9] Y.K. Malaiya and J. Denton. Requirements volatility and defect density. In *Proc. of the 10th Int. Symp. on Software Reliability Engineering (ISSRE '99)*, pages 285–294. IEEE Computer Society, 1999.

[10] T. J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, 1976.

[11] K. W. Miller, L. J. Morell, R. E. Noonan, S. K. Park, D. M. Nicol, B. W. Murrill, and M. Voas. Estimating the probability of failure when testing reveals no failures. *IEEE Transactions on Software Engineering*, 18(1):33–43, 1992.

[12] G. J. Myers, C. Sandler, T. Badgett, and T. M. Thomas. *The Art of Software Testing, Second Edition*. Wiley, 2004.

[13] L. Nachmanson, M. Veanes, W. Schulte, N. Tillmann, and W. Grieskamp. Optimal strategies for testing nondeterministic systems. *SIGSOFT Software Engineering Notes*, 29(4):55–64, 2004.

[14] M. Newman. Software errors cost u.s. economy 59.5 billion annually, NIST assesses technical needs of industry to improve software-testing. Press Release, `http://www.nist.gov/public_affairs/releases/n02-10.htm`, 2002.

[15] Felix Redmill. Exploring risk-based testing and its implications. *Software Testing, Verification and Reliability*, 14(1):3–15, 2004.

[16] A. Saltelli, S. Tarantola, F. Campolongo, and M. Ratto. *Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models*. Halsted Press, 2004.

[17] H. Ural. Formal methods for test sequence generation. *Computer Communications*, 15(5):311–325, 1992.

[18] J. Voas, L. Morell, and K. Miller. Predicting where faults can hide from testing. *IEEE Software*, 8(2):41–48, 1991.