Extended Abstract: Getting Engaged

Simon Thompson

University of Kent, UK
s.j.thompson@kent.ac.uk

How do we engage with the artefacts that we build as software engineers or computer scientists? Programs are one thing: we can execute them and see how they behave, as well as engaging with them in other ways. Other artefacts – many of which are more abstract and formal – present more of a problem.

Mathematicians, who are used to dealing with rigorously presented ideas and proofs, have a social process for dealing with understanding. Arguably, however, they have just the same problem with a more formal presentation: for example, by all accounts Russell and Whitehead's *Principia Mathematica* is riddled with mistakes of minor and more major significance.

In the computing domain, what sort of questions do we want to ask about an artefact such as a *specification* or a *set of tests*?

- How do I know I've not made trivial mistakes?
- How do I avoid contradiction?
- And, most importantly, how do I know when I have written enough?

One answer is to make specifications *executable*, as Howard's work on MetateM did for temporal logic, allowing authors to play out the consequences of their logical descriptions, and, in the process, learning how to write complex temporal formulas as a benign side-effect.

Recent work with Thomas Arts and Pablo Lamela Seijas of Chalmers and Quviq AB [2, 1] looks at how we can *engage with test sets*, answering questions like those above.

This work comes out of the FP7 ProTest project on property-based testing using QuickCheck to test systems written in Erlang and C. In property-based testing users specify logical properties of systems, rather than unit tests: these properties are then tested at randomly-generated values. A major challenge for the take-up of property-based testing among users is to write the properties themselves (just as it was in model checking). So, we have explored ways to build tools which can help people develop properties on the basis of existing unit test sets.

Our work – which is presented in overview – looks at sets of tests for stateful systems. We show how to move from a set of unit tests to a state machine representing the 'simplest' implementation of the system. In itself, this tells us something about the scope and coverage of the tests, and can certainly be strong evidence that we haven't written enough tests. Even without an implementation we're able to generate an FSM in many cases, giving feedback on tests even before the system is implemented.

If we do have an implementation of the system under test we can then use the facilities of QuickCheck to generate further tests, so 'round tripping' from tests to state machine and back to an extended and improved test set.

References

[1] T. Arts, P. L. Seijas, and S. Thompson. Extracting quickcheck specifications from eunit test cases. In *Proceedings of the 10th ACM SIGPLAN workshop on Erlang*. ACM, 2011.

[2] T. Arts and S. Thompson. From test cases to FSMs: augmented test-driven development and property inference. In *Proceedings of the 9th ACM SIGPLAN workshop on Erlang*. ACM, 2010.