Obstacles and opportunities for model-based testing in an industrial software environment

Harry Robinson
Test Architect, Enterprise Management Division
Microsoft Corporation
harryr@microsoft.com

Abstract

The complexity of modern software projects is increasing, and traditional, hand-crafted test suites have become unwieldy and fragile. Model-based testing can provide a tremendous increase in testing capability, but modeling technology must be integrated into everyday software testing. Small-scale pilot projects, readily available tools and tester education have made the migration to test generation easier at Microsoft.

Introduction

Model-based software testing has been around since the mid-1970s, and modeling advocates have written many white papers which show its dramatic improvement over hand-crafted test automation. Yet only a small segment of the mainstream software industry currently uses model-based testing. Cultural change, lack of training and the need for readily-available test modeling tools are some of the causes of this slow growth.

Obstacles to Model-Based Testing

Software testers are often non-technical. In most companies, the test team is much less technical than the development team; often, they are people who "failed to meet the bar" for a development position. The limited technical background of this pool constrains the amount of test innovation that can be absorbed by the team.

Testing is viewed as a back-end, ad hoc activity. In most companies, testers are not involved until the product has been designed and coded. The many test outsourcing companies who provide testing services as a separate activity from the design cycle testify to this late involvement. Traditional testers therefore have little effect on software design, and are often unable to automate product testing.

Testers operate in a severe time crunch. Testers are constantly under pressure to produce bugs and test cases in the short term. They are discouraged from creating test generation systems that could be more cost-effective than traditional methods in the long run.

Formal requirements are rare. Advanced industries, such as telephony and avionics, have elaborate specifications. Most software development companies, however, have few written specifications; people work from natural language user scenarios that are inherently ambiguous.

Current test metrics do not map easily onto test generation. Common test metrics such as "Number of test cases created" lose their relevance when tests are generated by an automated system. Test managers are often unable to interpret the status of a model-based testing project if they rely on traditional metrics alone.

Test generation is not supported in the industry testing books or tools. Testers do not read academic journals. At best, they read books on testing and industry magazines. These sources have little to say on test generation, focusing instead on

test scripting for regression tests. Popular industry tools likewise support capture/replay and test scripting, not test generation. Managers are reluctant to invest in a method or a tool that does not have widespread industry and tool support.

Opportunities for Model-Based Testing

Modern software projects are overwhelming the traditional approach to testing. As product cycles shorten and applications become more complex, current testing methods are not finding the important bugs before release. Companies are paying attention to new technologies that can help them achieve the quality they need.

Testers are becoming more technical. The days of the non-technical tester are coming to a close. Companies are discovering that they need technically proficient engineers to handle software testing, and these engineers are more effective if they are involved early in the process. Engineers who are comfortable with advanced approaches are more open to trying test generation.

Test models can serve as executable specs. Natural language product specifications are too vague to provide guidance in testing. Often they are not updated to keep pace with the project. Test models describe how the system should behave and can serve as formal specifications for the desired behavior of the product. Because the tests are generated from this test model, the model/spec is kept up to date with the product.

Our Experiences

For the past several years, I have led a growing movement at Microsoft to replace manual and scripted testing methods with test generation techniques. Because Microsoft has been successful in the past with the traditional test approaches, introducing model-based testing into the Microsoft mainstream has been a struggle and a learning experience for all of us.

We started out with tiny model-based testing pilots and eventually graduated to full product team efforts and “best practice” status within Microsoft. We evangelized at the lowest and highest levels within the company. We created and taught courses. We published articles, gave talks inside and outside the company. We created internal tools to make migration simpler. By doing so, we provided an opportunity for product teams to fundamentally re-think their approach to software testing and quality.

A Lens for Looking at Innovations

Diffusion of Innovations [1] is a landmark book by Everett Rogers that analyzes how new ideas spread through a group of people. His approach provides a useful framework for the issues we faced introducing model-based testing at Microsoft.

Rogers lists five characteristics of innovations that can accelerate or impede adoption:

- **Relative advantage**: is your innovation better than the existing method?
- **Compatibility**: does your innovation integrate with the existing method?
- **Complexity**: is your innovation difficult to understand?
- **Trialability**: is it easy for people to experiment with your innovation?
- **Observability**: are the benefits of your innovation easily visible?

In the following sections, we will review how each of these characteristics affected our promotion of model-based testing at Microsoft.

Relative advantage

> "The greater the relative advantage of an innovation, the more rapid its rate of adoption is going to be." [2]

Is model-based testing more useful than the traditional system of hand-crafted test automation? Here we found that perceived advantage depended on the observer:

- Testers saw great advantage in having thousands of test sequences generated automatically, and they were doubly pleased to find that those tests could be updated by simple changes to the underlying model.
- Developers did not care about test case creation and maintenance, but were pleased that their code was being well-tested. Also, tests were available earlier in the cycle which stabilized their code earlier.
- Managers had the hardest time seeing the advantages of model-based testing. They could see that models were creating more tests than before, but it was harder to see if the tests were doing a better job testing the software. Test generation upsets existing metrics, such as test case count, that managers had previously relied on.

Given these perceptions, we promoted model-based testing in tiny, one-person pilots to the front line testers who could see its advantages in action. Over several rounds of testing and various projects, these testers slowly convinced their fellow testers, then developers and finally management that this approach was worthwhile despite what traditional metrics might say.

Compatibility

> "An idea that is not compatible with the prevalent values and norms of a social system will not be adopted as rapidly as an innovation that is compatible." [3]

Compatibility with traditional test systems is a major stumbling block for model-based testing. The new methods we were proposing were a giant step away from the old system – so much so that it was difficult to make sense of one system's metrics in the context of the other system.

We were asking people to think in a radically different way. People were used to writing tests by hand. We wanted them to write systems that would generate tests instead. Conventional metrics counted test cases; test generation made that measure irrelevant. People were used to storing tests in a test case manager; we asked them to generate tests on the fly.

Our compatibility strategy was to be flexible. This approach worked out to our advantage. We didn't go in claiming we had all the answers. We simply provided a technology that showed that it could achieve useful results and let the test teams decide how they would fit that technology into their process. Pilot groups felt good that we hadn't tried to force a methodology on them, and they often helped us address compatibility issues. Some groups preferred to keep test generation aside from their regular test cases, using it to find bugs that their regular tests could not find. Other groups chose to generate tests directly into their test case management system. Some groups began to question the wisdom of having a static test case management system at all.

Through conversations with people looking to bridge the compatibility gap, we learned why people might want to keep some test cases around. Some teams wanted to include test sequences that had found bugs in the past. Other teams wanted short, reproducible tests that could serve as a benchmark or as smoke tests. In these cases,

we agreed that it made sense to have ready-made tests sitting on the shelf rather than generating them each time.

We continue to work on compatibility issues. Our goal is to introduce model-based testing, not to eliminate existing test methods. As long as people are using test generation to provide better testing, it doesn't matter how they choose to make that test generation compatible with their existing architecture. In many cases, halfway measures were acceptable for the present. It remains to be seen what the final result will be.

Complexity

> "… new ideas that are simpler to understand will be adopted more rapidly than innovations that require the adopter to develop new skills ..." [4]

Model-based testing is a more complex technology than many software testers had ever used. Under the traditional testing paradigm, it was acceptable to hire people with little computer background to do software testing. Test generation, however, requires a good background in abstract thinking, in understanding the product and in understanding the development cycle. Many testers were stretched beyond what they were currently doing, sometimes far beyond their current skill set.

To lower this hurdle, we limited initial models to simple, easy-to-understand state machines. Then these state machines were used against features the testers knew well. We would take a test case the tester had written and show how models could generate hundreds of test cases from that single test case. These models were easy for people to understand and they often found significant bugs. By introducing modeling concepts gradually and using the models to find bugs early, we avoided much of the resistance that the complexity might initially have brought about.

Even so, testers faced with the complexity of creating models often resisted trying out the technique for the first time under a project deadline. We therefore gave frequent brown bag talks on model-based testing to test teams around the company. Microsoft Technical Education also began to introduce test generation concepts into the official software testing curriculum.

Trialability

> "New ideas that can be tried on the installment plan will generally be adopted more quickly than innovations that are not divisible." [5]

Front-line testers were the best evangelists we had, so we made it as easy as possible for them to experiment with model-based testing. We provided free internal model-based testing tools and sample models. The tools included 2-D and 3-D state graph displays that let the testers visualize the model they were creating.

To help teams get started modeling, we held two-hour “extreme modeling sessions." A model-based testing expert met with the test team to model a feature of their product. Testers were able to see the benefits of modeling without first having to learn the syntax of the modeling tools. This was not strictly “trialability” because it didn’t allow the testers to play around with the modeling system on their own, but it lowered the threshold for getting them to try it at a later time.

As we moved to larger and larger pilots, trialability became problematic because of existing infrastructures and metrics. As a test team allocated more resources, management began to realize that there were substantial disconnects between their familiar metrics and the metrics that would track progress in the model-based testing world.

Observability

> "The easier it is for individuals to see the results of an innovation, the more likely they are to adopt." [6]

How well can people see model-based testing succeed? The small pilots and the extreme modeling sessions helped us enormously in being observable. The ability of your system to find bugs carries enormous weight with testers. How could anyone deny the effectiveness of the model-based testing while watching the system crash?

Conventional metrics such as test case count and bug count proved to be either ineffective or misleading. One team came up with a creative solution to the observability issue. After they had been using test generation for several weeks, I asked how they were keeping their management informed of progress. They said that they didn’t try very hard to get their new testing to fit into the old metrics. Instead, they took their management past the test lab every few days to see the number of developers who were in there fixing bugs found using the new methods.

There is, however, a natural limit when using bugs as your metric for observability in model-based testing. As a team becomes more adept at modeling and reviewing

specs, they will find fewer crashing bugs. In fact, you will find fewer of any kinds of bugs because bugs will be stopped in the early stages. This lack of observability can be cured by educating managers to look beyond the bug count to what really matters. It is very important to put this education in place before the bug counts drop.

Conclusions

Model-based testing faces significant hurdles in an industrial software environment due to its relative complexity and its incompatibility with existing infrastructure. Introducing the technology through small pilots and providing education and tools has allowed the power of model-based testing to shine at Microsoft.

Model-based testing continues to gather momentum:

- More than 600 of Microsoft's 5000 testers are currently involved in some form of model-based testing.
- More than 35 Microsoft product teams are engaged in model-based testing efforts.
- Test Model Toolkit, our internal model-based testing tool, received the 2001 Microsoft Engineering Excellence Award.
- Microsoft Technical Education is now creating several model-based testing courses.

In short, model-based testing is becoming mainstream technology at Microsoft. To quote Craig Zhou, Director of Microsoft's Windows Test Infrastructure area:

> "As ease of use becomes increasingly important to customers, the complexity of generating complete test cases is also exploding and is a key challenge for test teams.
>
> A systematic and automatic way of defining the test matrix is needed everywhere, and that is exactly what model-based testing provides.
>
> Model-based testing will be the primary method of creating test plans and test cases."

[1] Rogers, Everett, Diffusion of Innovations, 3rd Edition, Collier Macmillan Publishers, 1983

[2] Ibid., p. 15

[3] Ibid., p. 15

[4] Ibid., p. 15

[5] Ibid., p. 15

[6] Ibid., p. 16