Chapter 9
Model-Driven Testing with Test Sheets

Michael Felderer
University of Innsbruck, Austria

Colin Atkinson
University of Mannheim, Germany

Florian Barth
University of Mannheim, Germany

Ruth Breu
University of Innsbruck, Austria

ABSTRACT

Test Sheets provide a new way of representing tests that combines the ease-of-use of tabular test description approaches with the expressiveness of programmatic approaches. Since they are semantically self-contained, and thus executable, they offer a more compact and easy-to-understand approach to test specification than code-level representations of tests. Nevertheless, since they still define tests at a relatively low-level of detail, they can be difficult to develop, and can become quite complex when describing large testing scenarios. Model driven testing approaches on the other hand support high level, graphical views of tests and provide methodological support for deriving tests from system models. However, they invariably rely on code to describe executable versions of tests, and tend to depict the different ingredients of tests in separated, isolated views. The development of test sheets is therefore likely to be significantly simplified by the support of a suitable model-driven testing approach, while model driven testing approaches are likely to be enhanced by the availability of compact, executable representations of tests in the form of tests sheets. In this chapter we therefore explore the synergy between test sheets and model-driven development in the context of the test stories methodology. This is an advanced standards-compliant method that covers the whole test development process from abstract requirements to concrete executable tests, but in the context of programming languages as implementation vehicles. In this chapter we present a case study in which we apply the test stories methodology using test sheets as the test description, execution, and reporting vehicle.

DOI: 10.4018/978-1-61350-438-3.ch009

INTRODUCTION

Model-driven testing has gained widespread acceptance in the last few years and today is not only an active research area but is also a regular component of mainstream software engineering projects. This was perhaps inevitable because the benefits of using high-level, (semi-) formal models to assist in the analysis and design of tests are similar to the benefits of using them for the analysis and design of application code. In fact, in the early stages of system specification the same models cover the behaviour of both because the specification of a software system describes the required behaviour of the main application code as well as the tests designed to check it. Most approaches for model-driven testing are therefore derived from mainstream modelling approaches such as the UML (e.g. UML2 Testing Profile (OMG, 2005), (Baker et al., 2007)) or SDL (e.g. TTCN-3 (Willcock et al., 2005)).

Although model-driven testing methods can significantly enhance the testing process, however, just like the models used to develop the main application functionality they always need to be mapped into code at the end of the development process. In other words, a model is still a means to an end rather than an end in itself, and cannot usually be used to drive tests without at least some code being written by hand. Another weakness of the models supported by today's model-driven testing approaches is that they are only able to capture one isolated aspect of a test and provide little support for an integrated view of how a system should behave. More specifically, the models used in today's model-driven development methods typically focus either on the test data (input and expected values), the execution logic or the result data. This makes them ideal for understanding key aspects of test while they are being analysed and designed, but less suitable for executing and documenting them.

Test sheets, on the other hand, have the opposite mix of strengths and weaknesses. They were designed to address the above problems by providing compact, executable description of tests that are nevertheless easy-to-write and understand. They are therefore comparable to test code in that they are executable, yet they are comparable to models in that they are relatively user friendly and platform independent. Nevertheless, like any executable specifications, writing test sheets is greatly simplified by the support of a high-level analysis and design models backed up by an accompanying methodology. The premise of this chapter, therefore, is that model-driven testing and test sheets are highly complementary, and that the latter providing an ideal vehicle for capturing the knowledge and insights gained in the former. The goal of this chapter is to present this potential synergy and illustrate how test sheets and model-driven testing complement each other. The model-driven development method that we use for the investigation is the Telling TestStories (TTS) methodology (Felderer, Breu et al., 2009). Although it primarily uses graph-based models, certain key views of the TTS approach are tabular.

However, the integration of a model-driven testing approach like TTS with test sheets has to fulfil several requirements:

- The approach supports the automatic validation and quality assurance of designed test cases.
- The approach emphasizes the uses of user-friendly models backed up by rigorous specification, validation and quality assurance techniques.
- The approach guarantees traceability between tests, requirements, system elements and the artefacts of the system under test.
- The approach has an operational semantics and a semantically self-contained tabular test notation.
- The approach has a self-contained semantics for all its artefacts.