Using UML to Partially Automate Generation of Scenario-Based Test Drivers

Jeremiah Wittevrongel, Frank Maurer
Department of Computer Science, The University of Calgary
Calgary, Canada

Abstract

Testing software is a time-consuming activity requiring a great deal of planning and resources to be effective. In many environments, the pressure to ship software quickly is overwhelming, and results in compressed development schedules. In this paper, we discuss a testing approach that supports developers in their task of creating automated functional test drivers for object-oriented software on a compressed schedule. We then discuss a tool designed and implemented to support the approach.

1 Introduction

The Internet has changed the pace of software development. Applications that were previously developed over the course of several years are now developed in months. Approaches like Extreme Programming attempt to drastically shorten development cycles and may be particularly well-suited to mid-sized software projects.

One of the cornerstones of Extreme Programming is the use of continual testing throughout the development process. A combination of unit tests and scenario tests form a continually evolving and always-available automated test suite.

For successful automated testing, support must come from both process and tools. SCENTOR [10] is an approach that aims to provide support for the generation of scenario-based tests using JUnit [7] as a basis. In doing so, it is hoped that creating scenario tests for software applications takes less time, leaving more time for running the tests and developing the system, in projects with a compressed schedule.

In Section 2, we motivate our work, and give some background. Section 3 illustrates how to derive test drivers from usage scenarios of a system. In Section 4, we discuss related work, and finally, Section 5 summarizes our results.

2 Motivation and background

Software projects often have huge time-to-market pressure; there is not always a lot of time for testing. Priority One under extreme time pressure should be ensuring typical use scenarios can be completed.

This suggests that testing of e-business applications should focus on user-visible functionality. In Extreme Programming, this user-visible functionality is described in user stories. SCENTOR adds Unified Modeling Language (UML) sequence diagrams on top of this approach, which serves to make the scenarios slightly more formal, and this allow partial automation for the creation of automated test drivers.

There is no reason this type of partial automation of test driver code could not be extended to generate test drivers from other types of UML diagrams (or ultimately a complete UML model), but our focus currently is supporting scenario-based testing. Tests based on typical-use scenarios ensure that the focus of testing is on the most-used parts of the system. This should be the focus under extreme time pressure.

The SCENTOR approach is also meant to support incremental testing. It should be possible to add test cases (individual tests with a single set of concrete parameter values and expected results) at any time to the existing test suites (collections of test drivers).

3 From scenarios to test drivers

As a proof of concept, SCENTOR was designed and implemented to support scenario-based testing of object-oriented software applications. SCENTOR is also targeted towards lightweight development processes that include only a partial set of UML models, while maintaining the Extreme Programming focus on the production of source code. Figure 1 shows the progression from scenarios to test drivers in SCENTOR.

Scenarios are first modeled as UML sequence diagrams, using a CASE tool such as Rational Rose. The UML model is then exported from the CASE Tool in the vendor-independent XMI (XML Metadata Interchange) format.

The developer would then load the XMI file into SCENTOR, and could optionally load a previously created test specification (in an XML format) as well (not shown in Figure 1).

Tests are specified based on UML sequence diagrams. A small suite of tests is typically based on the set of messages sent by a single object in a UML sequence diagram. The developer needs only to add concrete parameter values to each method call and specify the expected results. To preserve the focus on source code and avoid the need for developers to learn another language, a developer using SCENTOR enters these results in the underlying programming language (Java).

SCENTOR also helps with modular test design and maintenance with the ability to produce shared one-time setup code. This code is generated so that it is only executed once, regardless of the number of tests running.

The main benefit of sharing the setup code among a group of tests comes with reduced execution time for a large group of tests. Many test setup activities, like building or loading a preset database, take quite some time, and repeating this for each test means test drivers run much more slowly than if the setup is only done once for the entire test suite.

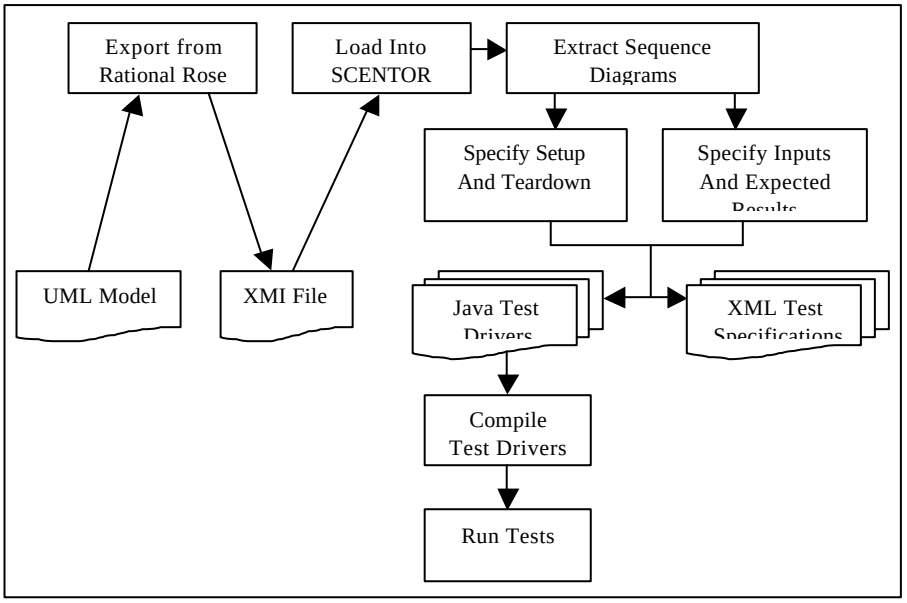

Figure 1: Moving from scenarios to test drivers

4 Related Work

SCENTOR takes some ideas concerning lightweight development processes from Extreme Programming discussed in [1], [2], and [9]. One of the most prevalent principles of Extreme Programming is the use of continual testing. SCENTOR aims to support continual scenario testing, but we assume an environment where UML is used to describe the scenarios. This is necessary to be able to partially automate test driver generation

JUnit [7], and other related tools, provide a simple framework for unit testing of software. Scenario tests can be specified using JUnit, but no specific support for scenario tests is included.

TOTEM [3] is a project that investigates ways in which UML diagrams can support derivation of test drivers for all levels of the system. In contrast with the lightweight UML modeling assumed by SCENTOR, however, TOTEM assumes a much greater use of UML in the development process, and a much more formal approach. TOTEM's approach, however, may not be feasible on a tight schedule.

Scenario-based testing of software in general is not a new idea. Discussions of scenario-based testing can be found in many published works, including [5], [6], and [8]. SCENTOR's contribution in this area is a framework that applies scenario-based testing to software projects that utilize some lightweight UML modeling during the development process.

SCENTOR assists developers in generating automated test drivers. The effectiveness of test automation in general, and the effectiveness of specific automation techniques has been the basis of much discussion, including [4] and [8].

5 Summary and future work

SCENTOR assists developers by forming a bridge between user scenarios and functional test drivers. By removing some of the repetitive, mechanical work required when creating automated test drivers, SCENTOR also aims to reduce the time required to develop them. Further, by supporting factored setup code explicitly, it is possible to realize a speed increase when executing large sets of tests. Test execution speed is crucial in a framework where regression testing is a on-going activity (as in Extreme Programming).

Future plans include an empirical evaluation of SCENTOR's effectiveness in reducing the time required for automated test development. This is the next step for the SCENTOR project.

In addition, there are other plans for the future of SCENTOR. SCENTOR currently has no facility for displaying the imported UML sequence diagrams graphically. Such an addition to the user interface would allow for much more intuitive selection of scenarios to use as the basis for test cases.

A third direction for the future of SCENTOR would be the addition of the ability to generate test drivers based partly on other types of UML diagrams as well. The XMI specification allows for an entire model to be represented as a document, and there are definite possibilities for using other information present in the model for the purpose of test driver generation.

References

1. K. Beck, "Embracing Change with Extreme Programming", Computer, IEEE CS Press, Los Alamitos, Calif., vol. 32, no. 10, Oct. 1999, pp. 70-77.
2. K. Beck, Extreme Programming Explained: Embrace Change, Addison Wesley, Reading, Mass., 1999.
3. L. C. Briand, Testing of Object-oriented software sysTEms with the Unified Modeling Language (UML), http://www.sce.carleton.ca/faculty/briand/totem/totem.htm (current Feb. 10, 2001).
4. M. Fewster and D. Graham, Software Test Automation: Effective use of Test Execution Tools, Addison Wesley, Harlow, England, 1999.
5. M. R. Lyu, ed., Handbook of Software Reliability Engineering, IEEE CS Press, Los Alamitos, Calif., 1996.
6. B. Marick, The Craft of Software Testing, Prentice Hall, Englewood Cliffs, N.J., 1995.
7. Object Mentor, Inc., JUnit, Testing Resources for Extreme Programming, http://www.junit.org/ (current 10 Feb. 2001).
8. W. E. Perry, Effective Methods for Software Testing, Second ed., John Wiley & Sons, New York, 2000.
9. J. D. Wells, Extreme Programming: A Gentle Introduction, http://www.extremeprogramming.org (current 10 Feb. 2001).
10. J. Wittevrongel, The SCENTOR Web Site, http://sern.ucalgary.ca/~milos/projects/scentor_intro.htm (current 10 Feb. 2001).