Generating test scenarios and improving software quality with model-based testing*

Caroline C. Letizio
Institute of Computing – Unicamp
Campinas SP, Brazil
caroline.letizio@gmail.com

Fernando R. Villas-Boas
Sofist – Intelligent Software Testing
Campinas SP, Brazil
fernando.villasboas@sofist.com.br

Bruno T. de Abreu†
Sofist – Intelligent Software Testing
Campinas SP, Brazil
bruno.abreu@sofist.com.br

Abstract

Today's challenge of software's lifecycle is to do more with less resources and costs. In software testing, automated test is being seen as a means to accomplish this goal. However, there is a lack of proposals that focus on test design automation. One way to do it and that is becoming common is model-based tests. This work briefly presents a tool that eases test design and that automatically generates test scenarios, presenting too the opinions of those who tried it.

1. Introduction

In the beginnings of software testing, test cases had to be designed manually, with the following inconveniences:

- Too many test cases to be manually designed;
- Intensive rework due to the huge number of test cases;
- Test team becoming unmotivated;
- Some errors when designing test cases;

One alternative that testers found to this problem was to automate tests in order to minimize time and effort without affecting test quality [2], and model-based tests has been proving to be an efficient way of automating.

One way to abstract the software's main idea and behavior is by modeling, either using a decision tree, a finite state machine, a case diagram, an activity diagram, or any equivalent representation. Testers noted that it is possible with these models to create scenarios and test cases, since it is a way to verify the software decisions and all the paths that data can take.

A test model is obtained from software requirements. It must have an exact syntax and exact semantics, i.e., the model must be executable by tools. Such tools should be able to create the model itself or to create tests from this model.

Presently there are many solutions both in the software industry and in the academy that focus on automating the execution of white box and black box tests [8, 9, 10]. Model-based testing is a black box testing technique, so it is neither necessary to know the type of language or tool that was used to develop the software, nor its internal structure [6], so this type of testing has been very effective. The following are its main advantages.

- The generation of test cases begins earlier at the development cycle, helping to find failures right from the beginning and reducing the cost of bug correction;
- It allows automated generation of tests;
- It reduces test generation costs, since in frequently changing systems a tester only have to modify the model of the system and then quickly recreate his tests, as opposed to recreating them manually and originating accidental mistakes [7];
- The model allows automated exploratory testing at the software;
- It allows regression tests, which involve testing the modified program in order to establish confidence in the modifications [2].

Organizations want to test software in an adequate but quick and as thorough as possible way [2]. This paper shows model-based testing in practice, and it is organized as follows: Section 2 describes a tool produced by Sofist, which generates tests scenarios from an activity diagram, while Section 3 presents some of the results of those tests in one of Sofist partners.

* The authors would like to thank the support of FAPESP given through research grant #2008/55297-0.
† Corresponding author

2. Model-based testing in practice

Since the source code of the components is generally not available in earlier stages of software development, we adopted a method to generate test cases from UML Activity Diagrams [3, 5]. Most of the diagrams are modeled using information taken from use cases, and this method was improved by Sofist R&D team. It resulted in a tool that generates test scenarios from one or many activity diagrams (AD). A test scenario could be defined as a high-level test case (one without detailed test input and output).

Test scenarios can serve as a guide to the test team, helping them to focus on what really needs to be tested. They can also be used as a way to generate many different test cases, because it is generic and does not tell which test data is needed as input; the test analyst must discover a way to execute that scenario. This last aspect makes test scenarios more flexible and useful even for agile development teams.

There are several advantages in using activity diagrams: 1) putting together information about the use case flow makes it easy to see what the software has to do, 2) it provides a higher understanding of the costumer about its business, since an AD is generally a workflow, and 3) it facilitates the identification of the impact due to changes at the use case flows.

The tool also enables the modeling of interaction among use cases. For instance, in unit testing and integration testing we can test each unit and later integrate them, until we have the complete system with all its units grouped. Furthermore, a standalone use case could be seen as a unit, therefore a scenario that describes interactions between two or more use cases can uncover more different types of faults than a scenario that has steps of just one use case [4].

Other benefits are the anticipation of the software's scenarios for tests, since its specification is usually available at the beginning of the development cycle. This anticipation reduces the risks of delay, in case the project has been made after the analysis or implementation stage, thus reducing costs even more. The use of ADs enables the advance identification of inconsistencies in use cases, or in the interaction between them, reducing the chance of occurring changes in requirements after the beginning of the implementation stage. A relevant issue is that 56% of fault findings in software, after it has been delivered, occur at the requirement stage [1].

3. Conclusions

Sofist is running a real-world proof of concept with one of its partners. The feedback from the partner's team is that many issues started to arise when the modeling was still ongoing, long before development. More than 150 issues in 25 use cases were raised so far, and all were relevant and valid according to the partners' business analysts.

This feedback validates our proposal, since we were able to anticipate the identification of issues that would impact not only the test team, but also the development team. Using Sofist's test design tool, we were able to generate automatically in less than 30 seconds more than 500 test scenarios that guaranteed 100% of coverage of 25 use cases' specification.

Another interesting feedback from the partner's team was the change in the team's mood. People that started doing test design based on models were happier, and when the authors asked the reason they got two answers: a) "Now I'm doing a thing that really excites me and puts my brain to work hard", and b) "I just can't wait to do a rework on just only a few models instead of 100 test cases inside a spreadsheet".

4. References

[1] D. Leffingwell, "Calculating your return on investment from more effective requirements management", Internet: http://www.ibm.com/developerworks/rational/library/347.html accessed February 12, 2011.

[2] E.R.C. de Almeida, B.T. de Abreu, R.A. Moraes, "Simple Approach to Automated Test Effort Estimation". In: 4th Latin-American Symposium on Dependable Computing, 2009, João Pessoa. Proc. of the Fourth Latin-American Symposium on Dependable Computing, 2009. v. 1. p. 1-2.

[3] I. R. D. C. Perez, E. Martins, "Automação em Projeto de Testes Usando Modelos UML". In Proc. of 1st Brazilian Workshop on Systematic and Automated Software Testing (SAST 2007). João Pessoa, PB, Brazil.

[4] L.C. Briand, Y Labiche, "A UML-Based Approach to System Testing". Software and System Modeling V. 1 N. 1 (2002).

[5] P. Zielczynski, "Traceability from Use Cases to Test Cases".Internet:http://www.ibm.com/developerworks/rational/library/04/r-3217/ accessed February 12, 2011.

[6] S.H. Edwards, "Black-Box Testing Using Flowgraphs: An Experimental Assessment", Software Testing, Verification and Reliability, Vol. 10, No. 4. (12 January 2001), pp. 249-262

[7] S.R. Dalal, A. Jain , N. Karunanithi , J.M. Leaton, G.C. Patton, B.M. Horowitz, "Model-Based Testing in Practice" ICSE 99 Proceedings of the 21st International Conference on Software Engineering. ACM New York NY, USA 1999

[8] Test Design Tools. Internet http://www.testingfaqs.org/t-design.html accessed March 18, 2011.

[9] Test Tools and Site Management Tools. Internet: http://www.softwareqatest.com/qatweb1.html accessed February 12, 2011.

[10] Testing tools and Unit testing tools. Internet: http://www.opensourcetesting.org accessed February 12, 2011.