A Novel Approach for Scenario-Based Test Case Generation

Baikuntha Narayan Biswal
Department of CSE
National Institute of Technology
Rourkela, India
baikunthanarayan@gmail.com

Pragyan Nanda
Department of CSE
National Institute of Technology
Rourkela, India
n.pragyan@gmail.com

Durga Prasad Mohapatra
Department of CSE
National Institute of Technology
Rourkela, India
durga@nitrkl.ac.in

Abstract: Testing of software is a time-consuming activity which requires a great deal of planning and resources. Model-based testing is gaining importance as a research issue. In scenario-based testing, test scenarios are used for generating test cases, test drivers etc. UML is widely used to describe analysis and design specifications of software development. UML models are important source of information for test case design. UML activity diagrams describe the realization of the operation in design phase and also support description of parallel activities and synchronization aspects involved in different activities perfectly. In this paper we generate test scenarios from activity diagrams, which achieve test adequacy criteria perfectly. Finally we generate test cases by analyzing the respective sequence and class diagrams of each scenario, which achieves maximum path coverage criteria. Also in our approach, the cost of test model creation is reduced as design is reused.

Keywords: UML-based testing, Scenario-based testing, Activity diagram, Sequence diagram, Class diagram, Test scenario, Test case.

I. Introduction

Model Based Testing (MBT) is gaining its popularity in both academia and in industry. As systems are increasing in complexity, more systems perform mission-critical functions, and dependability requirements such as safety, reliability, availability, and security are vital to the users of these systems. The competitive marketplace is forcing companies to define or adopt new approaches to reduce the time-to-market as well as the development cost of these critical systems. Much focus has been placed on front-end development efforts, not realizing that testing accounts for 50 to 75 percent of the lifetime development and maintenance costs [11,12]. Testing is traditionally performed at the end of development, but market-driven schedules often force organizations to release products before they are adequately tested. Model-based development tools are increasing in use because they provide tangible benefits by supporting simulation and code generation, in addition to the traditional design and analysis activities. These tools help users develop requirement and design models of target systems. The key challenge is to translate development oriented modeling languages into a form that is suitable for automated test vector generation, specification based test coverage analysis, requirement to test traceability, and design-to-test traceability.

Testing in industrial projects can be effective only when the testing effort is "affordable"; this means that the testing approach should be able to produce a test plan soon, and even when the software system is only partially modeled. Another important aspect in industrial testing is accuracy. Since inaccuracy can strongly diminish the testing utility, the best has to be done in order to enrich the testing results. In this paper, we focus on model-based testing. The term model-based testing refers to test case derivation from a model representing software behavior. Such a model may be generated from a formal specification such as, Z – specification, OCL etc. [1,4,10] or may be designed by software engineers through diagrammatic tools [3,5].

The paper is organized as follows: Section 2 gives a brief idea about the background and concepts we will be using in rest of the paper. Section 3 presents our approach which can be considered new, with respect to existing approaches. Section 4 shows some of the implementation results. Section 5 gives the comparison with the existing approaches. Section 6 concludes this paper and draws future work directions.

II. Background

Scenario-based testing is a software testing activity that uses scenario tests, or simply scenarios, which are based on a hypothetical story to help a person

978-0-7695-3513-5/08 $25.00 © 2008 IEEE
DOI 10.1109/ICIT.2008.43

think through a complex problem or system. They can be as simple as a diagram for a testing environment or they could be a description written in prose. These tests are usually different from test cases in that test cases are single steps and scenarios cover a number of steps. Scenarios are also useful to connect to documented software requirements, especially requirements modeled with use cases. Within the Rational Unified Process, a scenario is an instantiation of a use case (take a specific path through the model, assigning specific values to each variable). More complex tests are built up by designing a test that runs through a series of use cases.

Scenario testing works best for complex transactions or events, for studying end-to-end delivery of the benefits of the program, for exploring how the program will work in the hands of an experienced user, and for developing more persuasive variations of bugs found using other approaches.

III. TC-ASEC: The Proposed Approach

In this section we propose an approach TC-ASEC to generate test cases from design models using activity diagram, sequence diagram and class diagram. In the proposed scheme we are using gray-box testing method, where the advantages of both black-box and white-box testing are combined together. The generated test cases extends the logical coverage criteria of white-box testing and finds all possible paths from the design model which describes the expected behavior of an operation. We have extended our existing approach proposed in to generate test cases from design models. In our approach we had used activity diagrams as test models. First of all our approach parses the activity diagram and generates the test scenarios which satisfy the path coverage criteria. As activity diagrams represent the implementation of an operation like the flow chart of code implementation and an executing path is a possible execution trace of a thread of a program, the executing paths are derived directly from the activity diagrams. We have considered path coverage in our approach, since it has the highest priority among all the coverage criteria for testing. Our approach also handles the complicacy of nested fork joins using a criterion that checks whether the target activity state of a transition is a fork or an activity state. If the target of the transition is a fork, then the fork has higher priority over the activity state. So it should be considered first and then only the other path is considered. As a result of this priority criterion the complicated nested fork-join pair is handled properly in our approach. After all the possible test scenarios are generated we generate the corresponding sequence diagram, and class diagrams for each scenario. Now using category partition method we analyze the functional requirements to divide the analyzed system in functional units. To be separately tested. For each defined functional unit, the environment conditions (system characteristic of a certain functional unit) and the parameters (explicit input of the same unit) relevant for testing must be identified. Then test cases are then derived by finding significant values of environment conditions and parameters. The approach is described in Fig. 1.

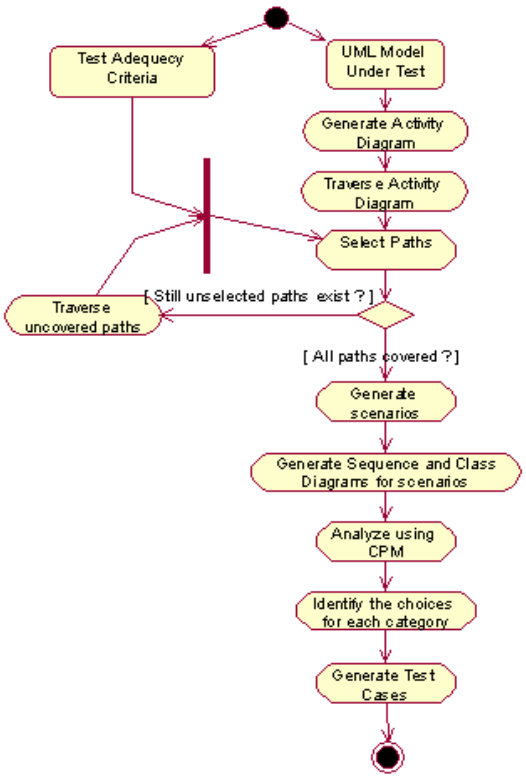

Fig. 1 Activity Diagram of our proposed approach

Test Scenario generation: TSAD

In order to generate test scenarios from activity diagram, we have considered all the activities, decisions, forks and joins as nodes. Our approach traverses the activity diagram using modified depth first search (DFS) method. In order to traverse the activity diagram from initial node to final node, our approach visits all the current nodes and the corresponding transitions released from the current node. Next, a record of the trace of a run of

the executing path of activity diagram is maintained by recording the visiting trace of the current nodes and transitions.

Each loop present in the activity diagram is executed at most once covering the corresponding activity states and transitions. A loop is bypassed in the sequence if it is already considered earlier. We have proposed an approach to generate test scenarios from design models.

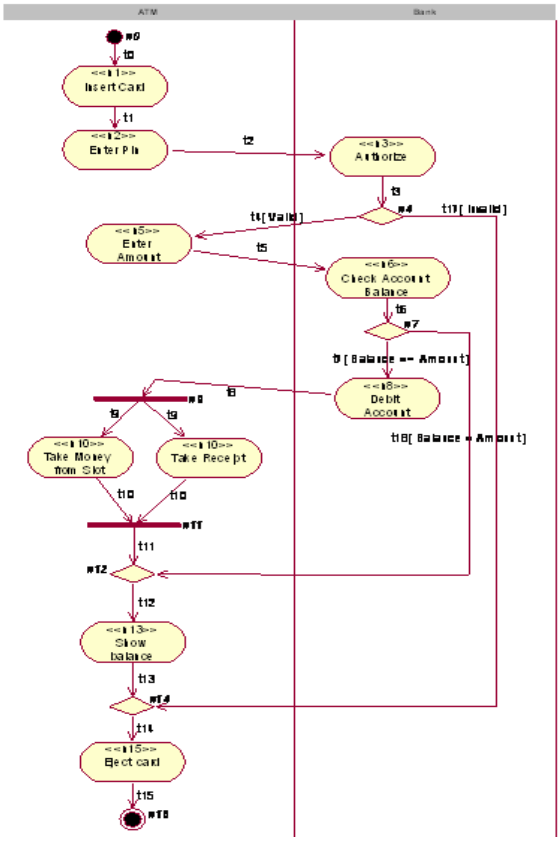

Fig. 2. Activity Diagram for ATM Withdrawal.

Test Case Generation

After all the test scenarios are generated, we analyze the corresponding sequence diagram for each selected scenario. Each sequence diagram is composed of objects and the messages they exchange. The objects involved in the Diagram are those that realize and execute the functionality described in the scenario through elaboration and message exchanges. In this phase class diagrams are also considered as a class diagram defines operations and attributes required for by classes for the interactions of their objects.

In our approach we have implemented category- partition method on sequence diagram and class diagram for generating test cases. The major steps involved are:

1. Analysis of sequence and class diagrams involved in the selected test scenario

2. Test Unit definition. Each object inside a sequence diagram is considered a Test Unit, since it can be separately tested and it represents and defines a possible use of system.

3. Search of setting and interaction categories. Interaction categories are the interactions that an object has with other objects involved in a same sequence diagram. Settings categories are attributes of a class (and corresponding sequence diagram's object), like input parameters used in messages or data structures.

4. Test Case construction. After both the categories are identified for each test unit significant values were chosen. For each found category its possible values and constraints are generated. For this purpose class diagram is used, where a preliminary description of a method implementation, its possible input values or the description of an attribute used and its significant values are found. By considering all the potential combinations of compatible choices, we derive the test cases. Finally for each test scenario, all the possible test cases are generated.

IV. Implementation and Results

This section discusses the results obtained by implementing the proposed Approach. We have implemented the complete approach using JAVA Swing and Rational Rose Version 7.0. We have implemented our approach taking ATM (Automatic Teller Machine) as the Case Study.

By implementing the approach we obtained 9 test scenarios. One of the test scenarios is as follows: TS: (a0) t0 (a1) t1 (a2) t2 [invalid] t3 (a3) [resolved] t5 (a4) t6t7 ((amt <= max) and (amt <= Bal) and ((amt mod 100) = 0)) t8 (a5) t9 t11 t7 (a6) t11 t12 (a7) t13 (a9).

After obtaining the scenarios we generated the sequence diagram and class diagrams for each test scenario and the two categories. By identifying significant values for each of the categories, we have obtained the final test cases. Both the positive and the negative test cases are generated for each of the generated test scenario using boundary-value analysis. Some of the generated test cases are given in Table-1. Further the test cases are analyzed for path coverage.

Table-1 Final Test Cases

TC-ID#	Test scenario	PIN	Amount Entered	Amount in Account	Amount in ATM	Expected Result
1	Successful withdraw	4987	200	1000.00	5000.00	Success Account updated
2	ATM Out of Money	4987	200	800.00	0.00	Withdraw Option Invalid
3	Insufficient Funds in ATM	4987	200	800.00	100.00	Warning Message Enter Amount
4	Amount is not Multiple of 100	4987	50	800.00	5000.00	Enter Amount
5	Incorrect Pin(= 0 try left)	4978	n/a	800.00	5000.00	Warning message Card Retained or not

V. Conclusion and Future Work

We generate test cases directly from UML behavioral diagram, where the design is reused. By using our approach defects in the design model can be detected during the analysis of the model itself. So, the defects can be removed as early as possible, thus reducing the cost of defect removal. First we generate test scenarios from the activity diagram and then for each scenario the corresponding sequence and class diagrams are generated. After that we analyze the sequence diagram to find the interaction categories and then use the class diagrams to find the settings categories. After analyzing each category, its significant values and constraints are generated and respective test cases are derived. The major advantage of our approach is that it handles the complicacy of nested fork-join pair which is more often overlooked by other approaches. It overcomes the limitations of the existing approach such as nested fork-join and loops. Test coverage criteria achieved is another advantage of our approach.

This approach can further be extended by generating test cases for the complete system i.e. by implementing the approach for integration testing as interactions between different components can be obtained from sequence diagrams. Also test drivers and test oracles can be generated for the proposed approach that will support developers in their task of creating automated functional test cases for object-oriented software on a compressed schedule. Moreover the overall approach is not fully automated. An automated tool can be developed for the proposed approach. The ultimate goal will be to address testability, coverage criteria and automation issues, in order to fully support system testing activities.

REFERENCES

1. Orest Pilskalns , Anneliese Andrews , Andrew Knight , Sudipto Ghosh , Robert France , Testing UML designs, doi:10.1016/j.infsof.2006.10.002
2. Jean Hartmann, Marlon Vieira, Herb Foster, Axel Ruder, UML-based Test Generation and Execution, Siemens Corporate Research, Inc.
3. Lionel Briand, Yvan Labiche, A UML-Based Approach to System Testing, Initial submission: 25 February 2002/ Revised submission: 20 June 2002 Published online: 12 September2002 – Springer-Verlag 2002
4. C. Canevet, S. Gilmore, J. Hillston, L. Kloul and P. Stevens, Analyzing UML 2.0 activity diagrams in the software performance engineering process, *WOSP'04* January 14-16, 2004, Redwood City, California.
5. Chen Mingsong, Qiu Xiaokang, and Li Xuandong, Automatic Test Case Generation for UML Activity Diagrams, *AST'06,* May 23, 2006, Shanghai, China.
6. Wang Linzhang, Yuan Jiesong, Yu Xiaofeng, Hu Jun, Li Xuandong and Zheng Guoliang, Generating Test Cases from UML Activity Diagram based on Gray-Box Method, Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04).
7. F.Basanieri, A.Bertolino, and E.Marchetti. The cow suit approach to planning and deriving test suites in UML projects. In proceedings of Fifth International Conference on the UML, LNCS, volume 2460, page 383397, Dresden, Germany, October 2002. Springer-Verlag GmbH.
8. F.Frankin and T.Leonhardt. SeDiTeC-testing based on sequence diagrams. In Proceedings 17th IEEE International Conference on Automated Software Engineering, pages 261-266. IEEE Computer Society, September 2002.
9. A.Bertolino and F.Basanieri. A practical approach to UML-based derivation of integration tests. In Proceedings of the 4th International Software Quality Week Europe and International Internet Quality Week Europe, Brussels, Belgium, 2000. QWE.
10. T.J. Ostrand and M.J.Balcer. The category-partition method for specifying and generating functional tests. Communications of the ACM, 31(6), June 1998
11. .Gourlay, J.S., Introduction to the Formal Treatment of Testing, Software Validation. Proceeding of the Symposium on Software Validation, 1983
12. Beizer, B. Software Testing Techniques, New York, New York: Van Nostrand Reinhold, 1983.
13. Graubmann, P., and E. Rudolph, HyperMSCs and Sequence Diagrams for use case modeling and testing, Proc. UML 2000 LNCS Vol.1939, 2000, Pages 32-46
14. Wittevrongel, J., and F. Maurer, Using UML to Partially Automate Generation of Scenario-Based Test Drivers, OOIS 2001, Springer, 2001.
15. P. Nanda, Dr. D. P. Mohapatra and S. K. Swain, Generation of Test Scenarios Using Activity Diagram, In Proceedings of SPIT-IEEE Colloquium and International Conference, Mumbai, India,vol-4, pages 69-73, February 2008.