AUTOMATIC TEST CASE GENERATION USING UML MODELS

[1]SAGARKUMAR P. JAIN, [2]KHUSHBOO S. LALWANI, [3]NIKITA K. MAHAJAN, [4]BHAGYASHREE J. GADEKAR

[1,2,3,4]Department of Computer Engineering, Pune University
E-mail: [1]jainsagar7@gmail.com, [2]khushbulalwani8@gmail.com, [3]nikitaskm@gmail.com,[4]gadekarbhagya@gmail.com

Abstract- We present a test case generation technique using UML models. We use the features of UML activity diagram and use case diagram which include the objects and their activities. All the conditions, messages, are included in the test cases. Our model derives test cases using full predicate coverage criteria. We construct Concurrent Control Flow Graph (CCFG) from activity diagram. We focus on generating test cases on sequence of action and the actors performing these actions. The generated test cases from our system can be used for integration and system testing. The test cases generated are suitable for detecting object interaction.

Keywords- activity diagram, full predicate coverage criteria, UML, use case.

I. INTRODUCTION

Software testing plays a crucial step in assuring software quality. It serves the purpose of quality assurance, verification and validation. Studies indicate that 50% of cost for software development is given to testing [1]. Generation of test cases is a time consuming process. Hence, automation is an important issue. There are two main approaches to generate test cases. One approach is generating test cases from requirements and design specification. The second approach is using source code [2]. Most of the requirements and high-level designs of software are documented in the form of models. These models can be used for test case generation of the system by automated means.

UML diagrams may be decomposed into several interconnected interaction diagrams. For generating test sequence, Activity diagram is alone not enough to decide the different components, i.e., input, expected output, etc. Hence we need use cases diagrams to realize actor.

The paper starts section II as an introduction to the basic concepts like UML diagrams used, CCFP, etc. The III section covers a literature survey of existing systems. The IV section discusses the proposed system. In this section, we build graphs, find test sequences and generate test cases in tabular form. Section V concludes the paper.

II. BASIC CONCEPTS

UML Models: 1. Use case:
A use case is an abstraction of a system response to external inputs. It accomplishes a task that is important from user's point of view. Thus, a use case focuses on only those features visible at the external interfaces of a system. A use case instance defines particular input values and expected results. Use case diagrams do not present architectural models, user-interface models or workflow models. Use case scenarios are usually not executed in arbitrary orders. Some use case scenarios need to be executed before others.

They may have <<extend>> and <<include>> dependencies as well as sequential dependencies which stem from the logic of the business process that the system supports. Hence, data comes in or goes out of the system through use cases. An actor gives input and receives output information through use cases.

Thus, we can have dependencies and constraints between use cases for each actor. So use case diagrams can be ornamented with extra ad-hoc information to show relationships and dependencies among use cases of a system [3]. Use case modeling is a different and complementary way of eliciting and documenting requirements.

The use case model provides a prime source for objects and classes. Use case consists of actors and roles.

An actor specifies a role that some external entity adopts when interacting with your system directly. "A specification of sequences of actions, including variant sequences and error sequences, that a system, subsystem or class can perform by interacting with outside actors."[4].

Use case lacks the following necessary elements of test design:

1) Domain Definition of each variable that participates in a use case.
2) Required input/output relationships among use case variables (called parameter).
3) Sequential Constraints among use cases.

The diagram below shows the use case diagram of one of the modules of Hospital Management System.

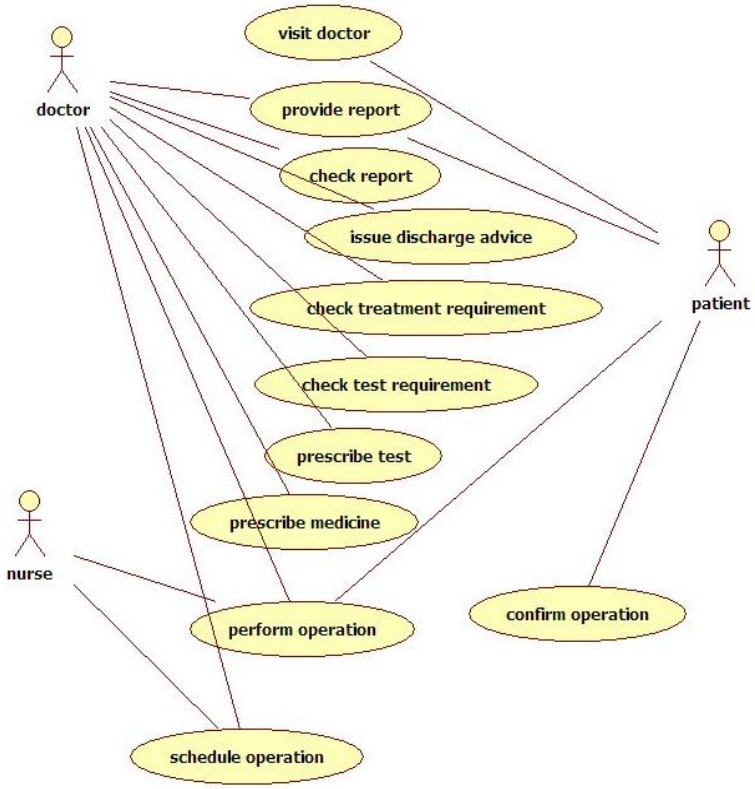


Figure 1. Use case Diagram of Hospital management system.

2. Activity diagram:

Activity diagrams are OO Flowcharts. Activity diagrams allow you to model a process as an activity that consists of a collection of nodes connected by edges .An activity can be attached to any modeling element for the purpose of modeling its behavior. The element provides the context for the activity, and the activity may refer to features of its context. Activities are typically attached to use cases, classes, interfaces, components, collaborations, operations. In our model we are attaching the activity diagram to use cases. Helps to identify which object does what activity [4]. Activity diagram shows the work flow of a system. The unique capability of activity diagrams is that they let you model a process without having to specify the static structure of classes and objects that realize that process. An example of activity diagram is as shown below:

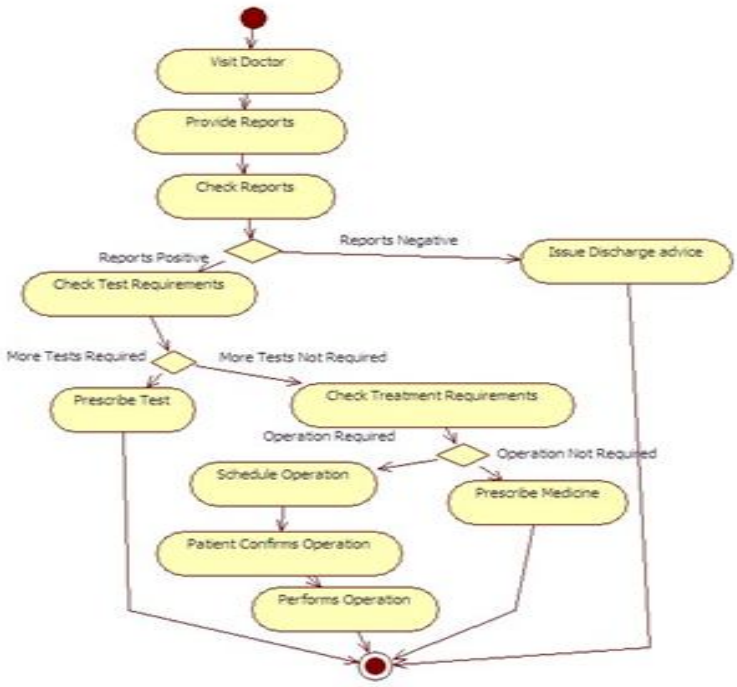


Figure 2. Activity Diagram of Hospital management system.

The above activity diagram is of Hospital Management System. It shows the work flow of the Hospital Management System.

Full-Predicate-Coverage-Criterion (FPC): A test set T satisfies the full predicate coverage criterion if and only if for each clause c in each condition in a sequence diagram there exist t1 in T such that t1 causes c to evaluate to TRUE and there exists t2 in T such that t2 causes c to evaluate to FALSE while all other clauses in the condition have values such that the value of the condition will always be the same as the clause under test [5]. This criterion requires all the predicates are checked i.e. all possible combinations of the different predicates in the condition are checked.

Concurrent Control Flow Path (CCFP): A CCFP is made by traversing from the initial node to the final node and concatenating all the nodes in the path. Similar to conventional CFP, special considerations regarding decision nodes should be made in terms of conditions and loops, i.e. two different CCFPs for true/false edges of a condition should be derived.

III. A LITERATURE SURVEY OF EXISTING SYSTEM

The approach proposed by Tonella et al. [6] generates test cases based on UML sequence diagrams that are reverse engineered from the code under test. Samuel et al. [7] presented a novel methodology for test case generation based on UML sequence diagrams. They create message dependence graphs (MDG) from UML sequence diagrams. Edge marking dynamic slicing method is applied on MDG to create slices. Based on the slice created with respect to each predicate on the sequence diagram, test data are generated. They have used a test adequacy criterion named slice coverage criterion.

They generate test cases for cluster/integration level testing. Sharma et al. [8] transformed a UML use case diagram into a graph called use case diagram graph (UDG) and sequence diagram into a graph called the sequence diagram graph (SDG) and then integrating UDG and SDG to form the System Testing Graph (STG). The STG is then traversed to generate test cases for system testing. They have used state-based transition path coverage criteria for test case generation. Nayak et.al. [9] Proposed an approach of synthesizing test data from the information embedded in model elements such as class diagrams and sequence diagrams.
They used OCL constraints. In their approach, they annotated a sequence diagram with attribute and constraint information derived from class diagram and OCL constraints and maps it onto a structured composite graph called SCG. The test specifications are then generated from SCG

IV. PROPOSED SYSTEM:

Our proposed approach for test case generation is based on the artifacts produced at the end of the analysis stage of the development. The artifacts include use case diagram and activity diagram for each use case. The approach requires the specification of such sequential constraints in the form of an activity diagram.

Our proposed model has the following steps to generate test cases:

1) Convert the activity diagram given to activity graph.
2) Extracting CCFP from Activity Graph.
3) Generate Test sequences from the activity graph.
4) Finally using the use case diagram and the sequences to generate the test cases.

Step 1:
In this graph, the vertices are the use cases or action nodes in the activity graph and the dependency edges are the edges between the activity action nodes. The fork and join nodes are deleted while constructing it. The Activity Graph of the activity diagram (Figure 2) of Hospital management is as follows.

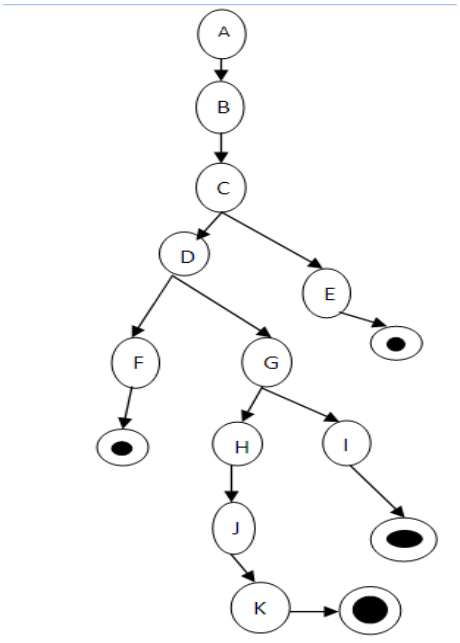


Figure 3. Activity Graph of corresponding Activity diagram in Figure 2.

Step 2:
Extracting CCFP from Activity Graph:
The next step is extracting Concurrent Control Flow Paths from the CCFG. All the different branches taken are identified and all possible concurrent paths are derived. If the number of branches is n, then the number of different possible paths can go up to 2*n. In the above example, we obtain five different paths. The different possible paths obtained are

1) A-B-C-E.
2) A-B-C-D-F.
3) A-B-C-D-G-I.
4) A-B-C-D-G-H-J-K.

Step 3:
To derive activity sequence Depth-First search is used.
The algorithm used to generate this sequence is:
Input: Activity Diagram
Output: Test Sequence
Let TS be Test Sequences.
Let PD be Parameterized Dependencies.
Step 1: Traverse the activity diagram using Depth-first Search.
Step 2: If loop present
Step 3: Make one iteration.
Step 4: End if.
Step 5: for each pair (A, B) of instantiated sequence in PD
Step 6: TS= Merge (A, B)
Step 7: Store all Test Sequences TS.
Step 8: End.

The test sequence generated is:

1) Visit doctor-Provide reports-Check reports-Issue discharge advice-end.
2) Visit doctor-provide-repots-check reports-check test requirements-prescribe test-end.
3) Visit doctor-provide reports-check reports-check test requirements-check treatment requirements-prescribe medicine-end.
4) Visit doctor-provide reports-check reports-check test requirements-check treatment requirements-schedule operation-patient confirms operation-perform operation-end.

The test sequence is also known as flow.

Step 4: The test sequence is displayed. Using the Use case Diagram the actors and their activities are determined and the test cases are generated in a tabular form. The table consists of the following information.

1) The actor /actors involved.
2) The action performed by the actor/actors.
3) Condition.
4) Next action to be performed.

The test cases in tabular form for the discussed example of hospital management system are as follows:

Actor	Action performed	Condition	Next action to be performed
patient	Visit doctor	Yes	Provide reports
Patient	Visit doctor	No	Error
Patient	Provide reports	Yes	Check reports
Patient	Provide reports	No	Error
Doctor	Check reports	Reports positive	Check test requirements
Doctor	Check reports	Reports negative	Issue discharge advice
Doctor	Check test requirements	More test required	Prescribe test
Doctor	Check test requirements	More test not required	Check treatment requirements
Doctor	Prescribe test	Yes	End
Doctor	Prescribe test	No	Error
Doctor	Check treatment requirements	Operation required	Schedule operation
Doctor	Check treatment requirements	Operation not required	Prescribe medicine
Doctor, nurse	Schedule operation	Yes	Confirm operation
Doctor, nurse	Schedule operation	No	Error
Patient	Confirm operation	Yes	Perform operation
Patient	Confirm operation	No	Error
Doctor, patient, nurse	Perform operation	Yes	End
Doctor, patient, nurse	Perform operation	No	Error

CONCLUSION

We have presented a strategy for integration testing which combines information from use case and activity diagram. Our approach exercises object interactions in the context of use case dependencies to fulfill the requirement of the user. The generated test cases may lead to the part of the test cases for system testing by combining it for all use case scenarios together without mine it separately. Our proposed technique uses XML, which is latest de-facto standard for exchanging UML models.

REFERENCES

[1] R.V. Binder. "Testing Object-Oriented Systems Models, Patterns And Tools". Object Technology Series. Addision Wesley, Reading, Massachusetts, October 1999.

[2] S. Muchnick, Advanced Compiler Design And Implementation, First Ed., Morgan Kaufmann, 1997.

[3] P.L. Navarro, D.S. Ruiz And G.M. Perez, "A Proposal For Automatic Testing Of Guis Based On Annotated Use Case Advances In Software Engineering.", Vol. 2010, Article ID 671284, Doi:10.1155/2010/671284, 2010.

[4] Jim Arlow, Ila Neustadt, "UML 2 And The Unified Process", Principles, Object-Oriented And Design, Second Edition, Pearson Education.

[5] S. Ghosh, R. France, C. Braganza, N. Kawane, A. Andrews, O.Pilskalns, "Yest Adequacy Assessment For UML Design Model Testing", In Proceedings Of 14th International Symposium In Software Reliability Engineering (ISSRE), pp.332,2003.

[6] Tonella, P., Potrich, A. "Reverse Engineering Of The Interaction Diagrams From C++ Code", In Proceeding Of IEEE International Conference On Software Maintenance, pp. 159-168,2003.

[7] Philip Samuel, Rajib Mall, "A Novel Test Case Design Technique Using Dynamic Slicing Of UML Sequence Diagrams" e-Informatica: Software Engineering Journal, Volume 2, Issue 1, 2008.

[8] M. Sarma, R. Mall, "Automatic Test Case Geneatin From UML Models",IEEE Computer Society, Pp.196-201, 2007.

[9] A. Nayak And D. Samanta, "Automatic Test Data Synthesis Using UML Sequence Diagrams." Journal Of Object Technology, Vol.09, No.2, Pp.75-104, March-April 2010.

★★★