A MODEL BASED TESTING APPROACH FOR GENERATION OF TEST CASES FROM PROBLEM STATEMENT USING UML ACTIVITY DIAGRAM

Upendra Verma, SVKM'S NMIMS University, India, (upendra.nmims@gmail.com)

R.K. Rambola, SVKM'S NMIMS University, India, (radhakrishna.rambola@nmims.edu)

Pratiksha Meshram, SVKM'S, NMIMS University, India, (pratiksha.meshram@nmims.edu)

ABSTRACT

Software Testing is an important activity used for quality assurance, reliability estimation, validation and verification of software products under development. There is a need for effective testing techniques for larger, complex and safety critical software applications in different fields of life such as defense, medical, automobiles and control systems for different products. Recent approaches have been taken by the researcher to use UML models for the purpose of software testing. Model-based testing is a software testing technique in which the test cases are derived from a model that describes the functional aspects of the system under test. Model based Testing in software engineering is gaining widespread importance due to faster and automatic generation of test suites for validating software systems. Model-based testing is a promising approach for achieving better fault detection. Testing using Models is a novel approach utilizing the key concepts of black-boxtesting.

In this paper we have generated test cases using activity diagram. This paper will make help to the researcher interested in the field of software testing using the UML model.

Keywords: Model based Testing, UML, Black Box Testing.

I. INTRODUCTION

In recent years, Model-Based testing (MBT) has unfolded into being an effective alternative for programming-based testing methods (Kyaw, A.A. and Min M.M, 2015. Model based testing is structured, comprehensive, accurate and measurable form of testing the software which is independent of the programming language and the platform used for building of any system (Kyaw, A.A. and Min M.M, 2015). Testing is one of the dominant approaches for determining the quality of systems. It is widely accepted in industries and academia (Swain, S.K., Pani.,S.K. and Mohapatra, D.P,2010) . But, manual testing is time consuming, labour-intensive and susceptible to errors (Nayak and Samanta ,12). Model-based testing technique has been adopted as an integrated part of the testing process. Commercial tools are developed to support model-based testing. Testing technique may be structural testing that is a code based approach or behavioural that is specification based approach for software testing. Structural testing is dependent on coding phase and being initiated as soon as the code becomes available. This type of testing detects the errors of commission after the implementation but unable to find error of omission. However, behavioral testing is specification based and being initiated as soon as the modeling artifacts become available in the initial stages of software development. This model based testing is capable to identify the error of omissions and expedite the software development because this type of testing activities works in parallel with coding phase immediately after the availability of modelling artifacts that are used for test case generations. Testing activity can be implemented any time in the development phase as the requirement becomes available. Traditional software testing considers only static view of code which is not sufficient to test dynamic behavior of object oriented software. Use of structural testing for object oriented software is complex and tedious task where model based testing helps software engineers to find test information with simple processing of models instead of code. Moreover, model-based testing promotes test case generation in parallel to coding during the development process that minimizes the developmentprocess.

II. LITRATURE SURVEY

Use Case Diagram is represent requirements in the form of interaction between system and its external actors that maybe another system or human user; depicting functional and behavioral capability of a system in user-centric perspective. These requirements are captured in natural languages for elicitation and specification of the system development. The building blocks of UCM are use case, scenario, and actors. Use cases and its scenarios help in conformance testing that is to determine whether a system meets its requirements or not (Nayak and Samanta ,12).

Proposed an approach for test case generation from use case diagram and sequence diagrams of the use cases. This approach follows three steps for test generations: constructing use case dependency sequences, finding test requirements from sequence diagrams and finally generating test cases. Use case dependency sequences, that depict the order of executions of use cases, are also produced in three steps: first activity diagrams for use cases of all actors in the use case diagram, then use case dependency graph (UDG) are formed for these activity diagrams and finally use case dependency sequences are generated using these UDGs. In this activity diagram, action nodes represent the use cases and the transition from one node to another represents the sequence of execution order. Similarly fork and join nodes in the activity diagram are used to represent any non-sequential (parallel) execution of use cases if exist in use case diagrams. Then in UDG, the vertices represent the action nodes and the edges of this graph between vertices represent the transitions of activity diagram but the fork and join are deleted in the UDG and its being replaced with multiple incoming and outgoing edges to vertices. For identifying use case dependency sequences the depth-first search is used on directed UDG. Sequence diagrams for all use cases in the use case diagrams are plotted using the features of UML2.0. Then test case sequence generation from these sequence diagrams are also done in three steps: first activity graph is formed from sequence diagrams using mapping rules, in second step concurrent control flow graph (CCFG) is formed from these activity graphs and in last third step the concurrent flow paths (CCFP) from these CCFG are created by identifying all possible paths from starting node to the end node of CCFG. These CCFP are assumed as sequence diagram test sequences and used in combination with the use case dependency sequence for test casegeneration.

(Nayak and Samanta ,12) also proposed an approach for test case generation from sequence diagrams. This technique also used the diagram frame of UML 2.0 for sequence diagram constructions what are mentioned as combined fragment for test case generations. These diagram frames represent different fragment or collection of messages passing in the sequence diagrams that alters the sequential flow of message passing in that interaction diagrams for a use case. Such combined fragments are used as assisting in generating the effective test case generation for the use cases having multiple scenarios modelled in the same interaction diagram using these combined fragments. This approach also enriches the sequence diagrams with attributes and constraint information using Object Constraint Language (OCL) from class diagrams. Sequence diagram is converted into a graph called Structured Composite Graph (SCG) and then test cases are generated from these SCGs. Structured composite graphs has two types of nodes which are worthy to be mentioned: Block node that represents a set of messages with same sequential flow and Control node that represents the branching or transfer of flow of message passing in the sequence diagram. There are four types of control nodes as decision node, merge node, fork and join nodes. The representations of decision and merge node are those combined fragments in UML 2.0 what depicts the conditional flows i.e. “alt” and “opt” frames and the representations of fork and join are those combined fragments in UML 2.0 that represents the parallel flows of message passing i.e. “par” frame in sequence diagram. As the SCG is constructed, these are being used for test case generations with depth-first searchmanner.

(M. Utting, B. Legeard,2015) proposed an approach for test case generation using generic algorithm for optimizing and validating test cases by prioritization. They used IBM product Rational Rose for constructing the sequence diagrams and used with .MDL extension. Necessary information for the test case generation are extracted from these sequence diagrams and then sequence dependency table is being created. Now test paths are generated from sequence dependency table (SDT) and a generic algorithm is adopted to implement to these test paths for test case generations.

(Kosindrdecha and Jirapun ,2015) identified some research challenges in finding test cases from requirements captures as use cases. They used a large number of use cases in requirement domain for identifying and ensuring the test of critical domain requirements and minimizing test case suite while maintaining the effectiveness of testing. To overcome these challenges, a new test generation technique is proposed with requirement prioritization and producing a comprehensive and effective test case suites. They add two more steps to waterfall process of development in the form of requirement prioritization and effective test case generation. Requirement prioritization is added with the intention to overcome the problem of large domain of requirements in large software systems. It has two sub-steps as classifying and then prioritizing requirements. This technique has the limitation to work on fully dressed use cases only. It works by extracting use case scenarios from fully dressed use cases, automatically producing test case scenarios from these use case scenarios, and the end producing test case information; input data is manually created, and then for it the expected and actual data is produced along with pass/fail result of the test case.

The research paper (Swain, S.K., Pani.,S.K. and Mohapatra, D.P,2010) describes a methodology of implementing MBT for testing and validating mobile phone applications. It discusses various advantages of using this technique in terms of automatic test case generation, error detection, scrutinized quality of tests, reduction in the cost and time due to evolution of models rather than program instructions. The authors also encountered certain challenges in terms of portability of test cases and requirement of expertise in the field of modelling the test cases using UML diagrams. Their study focused on Event Sequence Graph modelling technique to test Android applications. This paper in the

beginning imposed a question if the concepts of MBT could be used to validate the functional requirements of mobile apps, which later provided experimental results that MBT combined with Event sequence graphs can be used as an effective method to model and generate test suites for mobile apps developed on this platform.
A Research work (Swain, S.K., Pani.,S.K. and Mohapatra, D.P,2010) designed and generated test cases from UML state chart diagrams. In this Function Minimization Technique is applied and to generate test cases automatically from state chart machines. Firstly, a state chart diagram is constructed, and then it is traversed by depth first search algorithm to select predicates for initial data set. This technique achieves adequate test coverage without increasing the number of test cases. The research claims to achieve coverage in terms of State coverage, transition and transition pair coverage. The name given to the approach is AGeTeSC i.e. Automatically Generating Test cases from State Charts. It has been implemented using java and Net Beans IDE by taking Soft-drink Vending machine as a case study in a prototype tool named JUnit. RationalRosetoolisusedtoproduceUMLstatediagramsandrelatedartefactsforeffectivetestgeneration**.**
In a paper presented by (M. Utting, B. Legeard2015), the authors have used activity diagrams for the test generation paths in which a UML activity diagram is converted to XMI file and evaluated with cyclomatic complexity procedure. Then out of possible test cases generated, Optimization algorithm is used to select best test path. A Research study (M. Utting, B. Legeard,2015) discusses various algorithms to derive test cases from the graphs. Brute-Force, Greedy-set coverage and prefix-graph coverage techniques are explained. Better algorithms are designed to minimize the cost of test paths. Detailed analysis of Prime-Path coverage is done. The main focus is on the test requirements. They have compared three solutions for minimum test path problem.Out ofalltheapproaches,theprefixgraphcoveragesolutionseemspreferable.
Another paper (Swain, S.K., Pani.,S.K. and Mohapatra, D.P,10) analyzes and generates test suites from UML Interaction diagrams by using condition slicing approach. The approach is accomplished by construction of a UML interaction diagram. It is then converted to a Dependency Graph. The conditional predicates on Dependency graph are selected and computation of the slices is done corresponding to each conditional predicate. At the end Test cases are generated related to testing criteria specified. The approach is meant for the cluster level testing and thus no redundant test cases are generated. Since interaction diagrams are used so therefore interaction faults are detected during test phase.

Model based testing in Practice (Kyaw, A.A. and Min M.M, 2015) comprises of various advantages in testing arena. Most of the errors are due to inappropriate or incomplete models and programs. MBT ensures cost effectiveness toa large extent and extensive model coverage criteria. The study in this paper is confined to functional/operational testing. Several similarities and differences are specified between program verifiers and model based testing methods depicting the advantages and key points in this area. Model based methodology restricts the data types to be finite and small leading to extremely faster reasoning capability as compared to code based approaches. The support of model-based testing in industries will develop potential knowledge and skill in formal modelling and the operation of automated verification tools. Work presented in (Kyaw, A.A. and Min M.M,2015) specifies various advantages and techniques which can be used for effectively carrying out MBT in industries and at academic levels for generating fruitful test cases automatically, thus, saving lots of effort and cost spent during traditional white box testingstrategies.

III. UML ACTIVITYDIAGRAM

The Unified Modelling Language has several subsets of diagrams that it can model, including structure diagrams, interaction diagrams, and behaviour diagrams. Activity diagrams are a subset of the latter. Along with use case and state machine diagrams, they're used to describe business activities and software systems' functionality. You'll use a set of specialized symbols—including those for starting, ending, merging, or receiving steps in the flow—to build an activity diagram.

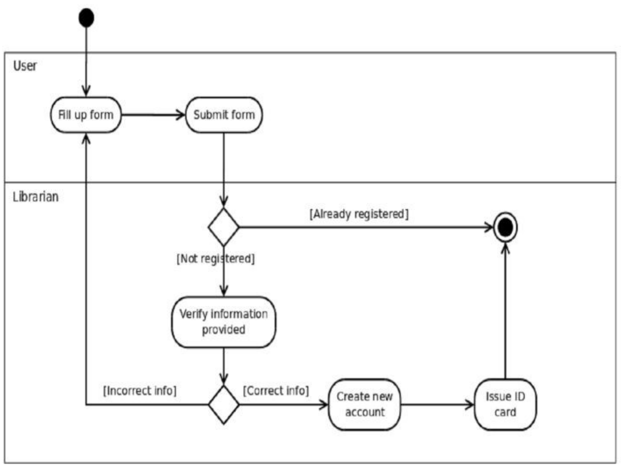
Stakeholders have many issues to manage, so it's important to communicate with clarity and brevity. Activity diagrams help people on the business and development sides of an organization come together. The activity diagram extracts the centre idea from Flowcharts(Kyaw, A.A. and Min M.M, 2015).

Use Cases for Activity Diagram
Activity diagrams have a number of benefits for any organization. Try using an activity diagram to: Demonstrate the logic of an algorithm. Describe the steps performed in a UML use case. Illustrate a business process or workflow between users and the system. Simplify and improve any process by clarifying complicated use cases. Model software architecture elements, such as method, function, andoperation.

Activity Diagram Components
Common components of an activity diagram include:
- Actions - a step in the activity wherein the users or software perform a giventask.
- Decisionnode-aconditionalbranchintheflowthatisrepresentedwithadiamond.Itincludesasingleinput and two or moreoutputs.
- Controlflows-thisisanothernamefortheconnectorsthatshowtheflowbetweenstepsinthediagram.
- Start node - symbolizes the beginning of the activity. This is represented with a blackcircle.
- End node - represents the final step in the activity. It's modelled with an outlined blackcircle.

Problem Statement for the User Registration:
A new user fills up the registration form for library membership (either online or in paper), and submits to the librarian. Of course, an already registered user can't create another account for him (or, her). For users' who don't have an account already and have submitted their registration forms, the librarian verifies the information provided, possibly against the central database used by the institution. If all information have been provided correctly, librarian goes on with creating a new account for the user . Otherwise, the user is asked to provide all and correct information in his (her) registration form. Once a new account has been created for the user, he (she) is being issued an ID card, which is to be provided for any future transaction in thelibrary.

Activity Diagram for User Registration:

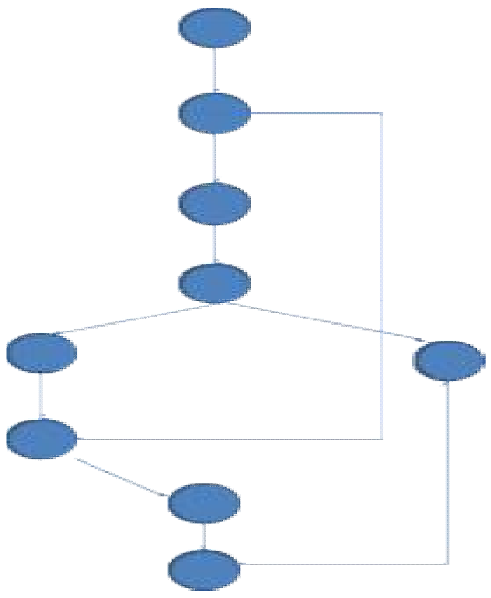


IV. DERIVE ACTIVITY GRAPH FROM ACTIVITYDIAGRAM

Test coverage indicates the extent to which a testing criterion such as path testing, branch testing or basis path testing is achieved. A popular test coverage criterion involves covering all the basis paths. Since basis path coverage is a stronger test coverage than transition or state coverage, it ensures that all edges visited at least once. So this will give us activity path coverage. These are finite as the numbers of basis paths are also finite.
A basic path is a sequence of activities where an activity in that path occurs exactly once. Note that a basic path considers a loop to be executed at most once.

Activity path coverage criterion:
Activity path coverage criterion, Activity Path Coverage is a sequence of non-concurrent activities (that is, activities which are not executed in parallel) from the start activity to an end activity, where each activity in the sequence has at most one occurrence except those activities that exist within a loop. Each activity in a loop may have at most two occurrences in the sequence. The aim is to use this coverage criterion for both loop testing and concurrency among activities of activity diagrams.
Before describing our new coverage criterion, we have mention about activity path and types of activity paths, namely (i) non-concurrent activity path, and (ii) concurrent activitypath.

Activity Flow Graph for the Activity Diagram of User Registration Problem:



Generate Test Path:

All possible test paths (test cases) are as follows:

Test Path 1: 1→2→3→4→9

Test Path 2: 1→2→3→4→5→6→7→8→9

Test Path3: 1→2→3→4→5→6→2→3→4→9

V. CONCLUSION

Test cases from UML Activity Diagram are generated and illustrated with the user registration procedure. Faults can be detected from the UML activity model during the analysis level. So, the time and cost of test case generation are reduced. We use activity coverage criteria in our approach. In our work we have used the UML 2.0 model features. The proposed model covers minimum activity nodes in the graph. Furthermore, the test cases are also reduced. The approach is not automated. In Future, We will develop automated tool for the proposed approach. This approach may be extended for other diagrams of UML for test case generation purpose.

I. REFERENCES

Kosindrdecha N.,and Daengdej (2010) .A test case generation process and technique. Journal of Software Engineering.

Jacobson. Basic Use Case Modeling (cont.) (1994)., Report on Object Analysis and Design. 1(3):7-9.

M. Utting, B. Legeard, Practical Model-Based Testing: A Tools Approach,Morgan Kaufmann Publishers Inc.,

San Farto., G.C. and Endo A.T. (2015). Evaluating Model based Testing Approach in the context of Mobile Applications. Electronic Notes in Theoretical Computer Science 314: 3–21.

Octaviano F. R., Thomazzo A. D., Camargo K., and Fabbri S. (2012). Test scenarios generation based on use cases, In CIbSE, 15-27).

Swain S. K., Mohapatra D. P., and Mall R.. (2010). Test case generation based on use case and sequence diagram, International Journal of Software Engineering.

Mark Utting (2008) . Position Paper: Model Based Testing.University of Waikato, New Zealand.

Nayak A. and Samanta D. (2010). Automatic test data synthesis using uml sequence diagrams, Journal of Object Technology, 09(2):5 – 104.

Ammann P. and Offutt J. (2008), Introduction to Software Testing.

Shanthi A.V.K. and Kumar G. M. (2012). Automated test cases generation from uml sequence diagram, International Conference on Software and Computer Applications, 41.

Dingel J., Schulte W., Ramos I., Abrahão S., and Insfran E. (2014), LNCS 8767 - Model-Driven Engineering Languages and Systems.

Swain, R., Panthi, V. and Mohapatra, D. P. (2012), Automatic Test Case Generation from UML State Chart Diagrams, International Journal of Computer Applications, 42(7):26 –36.

Abdurazik and Offutt J. (2000). Using UML collaboration diagrams for static checking and test generation. In Proceedings of the 3rd international Conference on the UML. Lecture Notes in Computer Science, Springer-Verlag GmbH, vol. 1939, pp. 383 - 395, York, U.K.

Kyaw, A.A. and Min M.M. (2015). An Efficient Approach for Model Based Test Path Generation.International Journal of Information and Education Technology, 5(10).

Swain, R., Panthi, V. and Bahera, P.K. (2012). Test Case Design using Slicing of UML interaction Diagram. Proceedings of 2nd International Conference on Communication, Computing & Security [ICCCS-2012], Elsevier.
Mall R. (2009) Fundamentals of Software Engineering. Prentice Hall of India, 3rd edition.

MarkUtting (2008).PositionPaper:ModelBasedTesting.UniversityofWaikato,NewZealand.

Kundu, D., & Samanta, D. (2009). A Novel Approach to Generate Test Cases from UML Activity Diagrams. Journal of Object Technology, 8(3):65-83.

Swain, S.K., Pani.,S.K. and Mohapatra, D.P. (2010). Model based object-oriented software testing, Journal of Theoretical and Applied Information Technology.