Test Cases Optimization Evaluation Using Efficient Algorithm with UML

Manoj Kumar, Prof.(Dr.) Mohammad Husain

Abstract- **The expenses of software testing is about 40-60% of the total cost of the software, so that reduction of test case numbers or test suite size is very much important and cannot avoid it without compromise the quality of the software. Error finding test case along with specific coverage criteria are more suitable for optimization that means the best one fit test case is selected above all and rest are ignored the number of test cases does not matter , they can be less or more either at the time of generation of test cases or after. For the reduction of test cases two options were proposed. One was at the time of generation and other was based on optimization concepts. The second case was preferred that means test case optimization after generation of the initial test case by random method. To reduce the test cases, the work was done on genetic algorithm [GA] based optimization approach.**

Index Terms- **Activity Diagram, Genetic Algorithm, Optimization, Test Cases, UML**

I. INTRODUCTION

Test case generation is the most important part of the testing efforts, the automation of specification based test case generation needs formal or semi-formal specifications. As a semi-formal modeling language, UML is widely used to describe analysis and design specifications by both academia and industry, thus UML models become the sources of test generation naturally. Test cases are usually generated from the requirement or the code while the design is seldom concerned [1]. UML is the most dominant standard language used in modeling the requirements [2, 3,4] and considered an important source of information for test case design. Therefore if it is satisfactorily exploited it will reduce testing cost and effort and at the same time improve the software quality.
Several researchers during the last decade have been using different UML models to generate test cases [5, 6, 7, 8, 9,10, 11]. Activity diagrams are one of the important UML models used in representing the workflows of stepwise activities and actions with support for choice, iteration and concurrency. Moreover, Activity diagrams can be utilized to describe the business and operational step-by-step workflows of components in a system [12].

Manuscript received September 09, 2013.
Manoj Kumar, Research Scholar, UPRTOU, Allahabad, India, (e-mail: iisemanoj@gmail.com).
Dr. Mohammad Husain, Director, Jahangirabad Institute of Technology, Barabanki, India, (e-mail: mohd.husain90@gmail.com).

It shows the overall flow of control between activities as well as the activity-based relationships among objects as it has all the characteristics that can improve the quality of the automatically generated test cases as well as using these test cases for system, integration, and regression testing [13].
Different sets of test cases used in those types of testing should have certain parameters or characteristics; they normally consist of a unique identifier, preconditions, a series of steps (also known as actions) to follow, input, and expected output, and sometimes post conditions [14]. Having this form still doesn't ensure that all test cases can be used and provide expected results as the quality of the generated test cases is the threshold. Quality of test cases depends on how far they would cover all the functionalities in a system under test [15, 16]. The test cases should be validated against some known quality standards [17, 18, 19] to ensure that they are in an acceptable form as well as ensure that they cover all the functionalities of a system.

Table 1: Generated Test Scenario

Test Scenario ID	Summary	Basic Scenario
01	To allow bank's customers to withdraw money from ATM machines anywhere in India	- Insert card - Input PIN - Select withdraw - Select a/c Type - Input balance - Get money - Get back card
02	To allow bank's customers to withdraw money from ATM machine in India	- Insert card - Input PIN - Select enquiry - Select a/c Type - Check balance - Select withdraw - Select a/c Type - Input balance - Get money - Get back card
03	To allow users to transfer money to other banks from all ATM machine anywhere in India.	- Insert card - Input PIN - Select transfer - Select bank - Select "To" a/c - Select a/c Type - Input amount - Get receipt - Get back card

Random test generation systems have been used to produce test cases; these systems tend to produce a uniform distribution of test case studies said that the testing was very much expensive and time taking process therefore automatic generation of test cases reduced the effort of a

tester and developer so cast and time. The optimal number of test cases required for testing was given by GA approach. That was the easiest flexible and could be applied to multi-objective optimization problems [20, 21].
Genetic algorithms [22] have been used to generate test sets automatically by searching the domain of the software for suitable values to satisfy a predefined testing criterion. These criteria have been set by the requirements for test data set adequacy of structural testing, such as obtaining full branch coverage and controlling the number of iterations of a conditional loop [23].

II. TEST CASE GENERATE FOR UML MODEL

A. Use of UML Activity Diagram to Generate Test Cases

Activity diagram is an important diagram among more than 10 diagrams supported by UML. It is used for business modeling, control and object flow modeling, complex operation modeling etc. Main advantage of this model is its simplicity and ease of understanding the flow of logic of the system. However, finding test information is critical task because of the following reasons [24]:

(a) Activity diagram presents concepts at a higher abstraction level compared to other diagrams like sequence diagrams, class diagrams and hence, activity diagram contains less information compared to others,

(b) Presence of loop and concurrent activities in the activity diagram results in path explosion, and practically, it is not feasible to consider all execution paths for testing. Here an approach is proposed for generating test cases using UML activity diagrams. In this approach, we consider a coverage criterion called activity path coverage criterion. Generated test suite following activity path coverage criterion aims to cover more faults like synchronization faults, loop faults.

B. Use of UML Sequence Diagram to Generate Test Case

After the generation of all the listed scenarios, corresponding analyzed sequence diagram for each scenario. Each diagram had object and they exchanged the message. The objects executed the functions given in the sequence diagram through elaboration and message exchange. Class diagram was very much important in this phase, diagram contained operations and attributes required for the interactions of their objects. Concerning this approach, the category partition method in the sequence diagram and class diagram for generating test cases is applied [25].

Table 2: General Test Scenario of ATM machine Generation

S.No.	Input	Steps	Expected Result
1	Withdraw Rs. 1000 from ATM machine	-take card - gather info - give Rs 1000 - return card	-Valid insertion -valid info -Rs. 1000 less -receipt & card slot empty
2	Single customer, 1 account Rs. 1000 withdrawal from checking account in Rs. 200 bills	-get deposit slip -swipe card -withdraw cash -print receipt	-filled correctly -valid info on screen -Rs. 1000 less - Balance & withdrawal amount

(a) The sequence and class diagrams[26,27] for identifying the various parameters and environments of the function, in selected test scenario are analyzed.
(b) Test Unit definition: Each object inside a sequence diagram considered as a Test Unit, since it can be separately tested and it represents and defines a possible use of system.
(c) Search of setting and interaction categories: Interaction categories are the interactions that an object has with other objects involved in the same sequence diagram. Settings categories were attributes of a class (and corresponding sequence diagram's object), like input parameters used in messages or data structures [25].

Table 3: Test Case Generation with Test Scenario

Test Case Transaction ID	**Test Scenario**	**Password**	**Withdraw Amount**	**Balance**	**Result**
1	Wrong Password(2 left)	3421	n/a	20000	Incorrect Password
2	Wrong Password(1 left	3422	1500	Wrong password	Message alert
3	Wrong Password(0 left	3412	n/a	20000	Warning message and card carried
4	Successful	3412	2000	18000	Successful Transaction

(d) Test Case Construction: After both the categories were identified for each test unit, significant values were chosen. For each found category, possible values and constraints were generated. For this determination, the class diagram is used, where an explorative of a method implementation and its possible input values (or the description of an attribute used and its significant values) are found. By considering all the possible combinations of well-matched choices, were derived the test cases. Finally, for each test scenario, all the possible test cases were generated.

III. GENETIC ALGORITHM BASED TESTING

The testing processes problems were considered as an optimization problem got solution of those problems with genetic algorithm. There was many different attributes for the testing process according to their optimization. For a type of problem there was an input domain and set of sequence all events were the input domains so each and every event had a fitness value. The event having more branches or decisions got more weighted value.
The value 1 was assigned for those transitions, which produced simple transition while value 0 was given for those who did not produce any transition. The weightage value 2 was given transition that produced branches or fork and joins.

- Initially for the given problem randomly a valid set of transition is selected.
- Secondly by using basic GA operations like selection, crossing over and mutation. New solution in the next generation is generated.
- Now, the fitness value of generations is calculated and then best fit test case selected.
- Process continued till reaching the stop condition as given by user.

Any successful test case was not the error proof. So error minimization technique minimize the percentage of errors is required.

IV. COMPARATIVE ANALYSIS

Four scenarios of ATM Banking system are seen. High priority scenarios considered at first instant and the process of calculating priority continued until all the scenarios of the system are covered. The four scenarios of the ATM Banking system are - ATM withdrawal, balance enquiry with receipt and PIN verification. The result obtained by our approach by considering the above said problems such as ATM withdrawal is also presented. Balance Enquiry in ATM and PIN verification in ATM system in, table 4.

Table 4: Experimental Results (Test Data Generated with Genetic Algorithm) -Comparison between GA Based Approach and Without GA Approach

Faults	No. of Fault Inserted(Tranition covered)	Faults are found without GA	Faults are found with GA
PIN	6	3	4
Balance	8	4	7
Withdrawal	11	6	8
Balance with receipt	13	9	13
Total	38	22	32

Fault Detection
Score = (Σ faults found / Σ faults injected) * 100
For Banking System tree node , 38 faults and 22 faults are injected found by without our approach and 32 faults are found by our approach, were revealed from the test cases generated. Using the above formula, 57.8% score is obtained without my approach for Bank system object diagram which showed optimization levels of our approach same formula repeated for my approach then got 71% score with my approach.
It was diagrammatically represented in the form of pie chart as shown in Figure 2-3. Below the figure, the analysis result produced by mutation testing

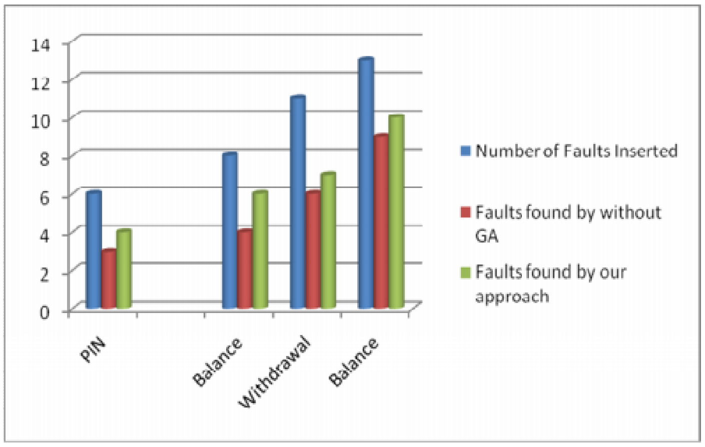

Fig. 1: Test Data Generated with GA

This approach uses genetic algorithm for generation of sub-optimal test cases using UML and we called this algorithm as genetic algorithm with UML (G-UML). Here, we use a constraint to satisfy the transition coverage as test adequacy criteria in genetic algorithm. The test adequacy criteria are all transition should be covered at least once, which is used as the stopping criterion for GA. So we called it as GA. But we have considered a special case in our approach by illuminating an ideal system like.

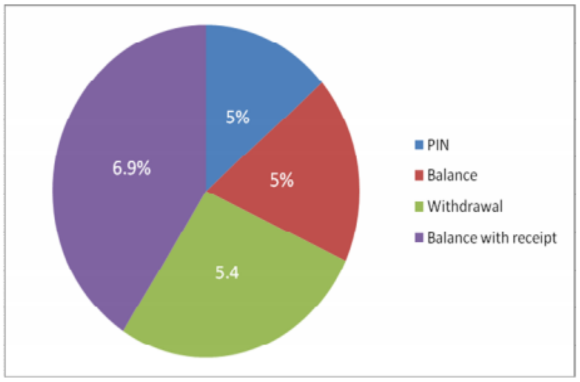

Fig. 2: Optimization Result Generated Without GA

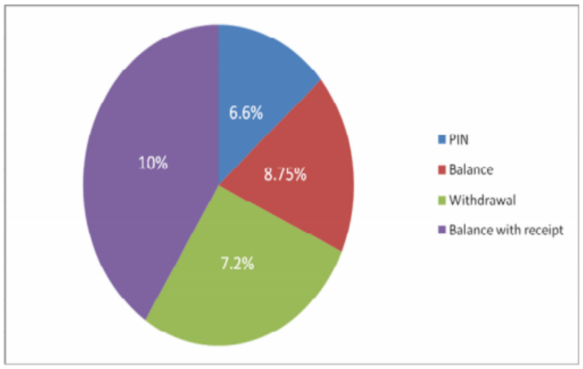

Fig. 3: Optimization Result Generated With GA

When the data using this approach, is implemented or tested, found the much optimized result. Without this approach the optimized result can't be found out. By comparing work with other approach much optimized solution is found. As shown in table 4 without using my approach, numbers of faults were inserted then optimized result is not found. But using my approach, if a number of faults were inserted then much optimized result is found. When a number of faults for PIN, Balance, Withdrawal and Balance with receipt are inserted, an optimization results for PIN -5%, Balance -5%, Withdrawal-5.4% and Balance with receipt-6.9 without my approach is found out. When my approach is implement then much optimize result for PIN -6.6%, Balance -8.75%, Withdrawal-7.2% and Balance with receipt-10% is found out.

V. CONCLUSION & FUTURE SCOPE

Here, we have focused on genetic algorithm in evaluation of object-oriented model. The problem of optimization is solved and increased the efficiency of a system. By this model, better memory management and code reusability is also facilitated. It may carry out towards the development of UML using genetic algorithms in future. This approach will help software developers to reduce their effort in generating test data before coding in order to create an effective and robust solution.

A genetic algorithm approach is used to obtain the sub-optimal (best fittest) test cases, which satisfied the test case adequacy criteria. This approach guaranteed the minimum presence of error, in the generated test case.

REFERENCES

[1] Wang Linzhang, Yuan Jiesong ; Yu Xiaofeng ; Hu Jun ; Li Xuandong ; Zheng Guoliang,Dept. of Comput. Sci. & Technol., Nanjing Univ., China ,Generating test cases from UML activity diagram based on Gray-box method,published in proceeding,APSEC '04 proceedings of the 11th Asia-Pacific Software Engineering Conference, pages 284-291 ,IEEE Computer Society Washington, DC, USA,2004.
[2] Pakinam N. Boghdady, Nagwa L. Badr, Mohamed Hashem and Mohamed F.Tolba, A Proposed Test Case Generation Technique Based on Activity Diagrams, International Journal of Engineering & Technology IJET-IJENS Vol: 11 No: 03, June 2011.
[3] G.D. Everett, R. McLeod, Jr., Software Testing: Testing across the Entire Software Development Life Cycle, IEEE press, John Wiley & Sons, Inc., Hoboken, New Jersey, 2007.
[4] B. Hasling, H. Goetz, K. Beetz, Model Based Testing of System Requirements using UML Use Case Models, Proceedings of the International Conference on Software Testing, Verification, and Validation, IEEE Computer Society Washington, DC, USA, 2008.
[5] M. Sarma, D. Kundu, R. Mall, Automatic Test Case Generation from UML Sequence Diagrams, Proceedings of the 15th International Conference on Advanced Computing and Communications, IEEE Computer Society Washington, DC, USA, 2007.
[6] S.K. Swain, D.P. Mohapatra, Test Case Generation from Behavioral UML Models,, International Journal of Computer Applications (IJCA) 6 (2010).
[7] H. Kim, S. Kang, J. Baik, I. Ko, Test Cases Generation from UML Activity Diagrams, Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD), Qingdao, China, 2007.
[8] M. Chen, P. Mishra, D. Kalita, Coverage-driven Automatic Test Generation for UML Activity Diagrams, Proceedings of the 18th ACM Great Lakes symposium on VLSI, Orlando, Florida, USA, 2008.
[9] C. Mingsong, Q. Xiaokang, L. Xuandong, Automatic Test Case Generation for UML Activity Diagrams, Proceedings of the international workshop on Automation of software test, New York, NY, USA, 2006.
[10] W. Linzhang, Y. Jiesong, Y. Xiaofeng, H. Jun, L. Xuandong, Z. Guoliang, Generating Test Cases from UML Activity Diagram based on Gray-Box Method, Proceedings of the 11th Asia-pacific Software Engineering Conference (ASPSEC), IEEE Computer Society, Washington, DC, USA, 2004.
[11] M. Chen, X. Qiu, W. Xu, L. Wang, J. Zhao, X. Li, UML Activity Diagram-Based Automatic Test Case Generation for Java Programs, The Computer Journal 52 (2009) 545-556.
[12] G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modeling Language User Guide, Addison-Wesley, 1999.
[13] R. Mall, Fundamentals of software engineering, 2nd ed. New Delhi: Prentice-Hall of India Ltd, 2008.
[14] D. Graham, E. Veenendaal, I. Evans, R. Black. Foundations of Software Testing ISTQB Certification, International Software testing Qualifications Board, 2010.
[15] L. Lazić, M. Medan. Software Quality Engineering versus Software Testing Process, The Telecommunication Forum TELFOR (Communication Forum), 2003.
[16] K. Smolander. Quality Standards in Business Software Development, Master of Science Thesis, Lappeenranta University of Technology, Department of Information Technology, 2009.
[17] IEEE standard for software test documentation, IEEE Std 829-1998, Published by Institute for Electrical and Electronics Engineers, New York.
[18] CMMI Product Team, CMMI for development v 1.3 (CMU/SEI-2010-TR-033), Carnegie Mellon University, Software Engineering Institute, 2010.
[19] S.J. Andriole, Software Validation, Verification, Testing and documentation, Petrocelli Books, Princeton, New Jersey, 1986.

[20] K. Deb, Multi-objective optimization using evolutionary algorithms,John Wiley & Sons, 2001.

[21] J.J. Durillo, A.J. Nebro, F. Luna, B. Dorronsoro, and E. Alba. jMetal:a java framework for developing multi-objective optimization metaheuristics.Technical Report ITI-2006-10, Departamento de Lenguajes yCiencias de la Computaci´on, University of M´alaga, E.T.S.I. Inform´atica,Campus de Teatinos, 2006.

[22] Tsoukalas, L., and Uhrig, R. Fuzzy and Neural Approaches in Engineering, Wiley, 1997.

[23] Jones, B.F. ; Dept. of Comput. Studies, Glamorgan Univ., Pontypridd, UK ; Sthamer, H.-H. ; Eyres, D.E.,Automatic structural testing using genetic algorithms, Software Engineering Journal, Volume 11, Issue 5, September 1996, p. 299 – 306,

[24] Puneet Patel and Nitin N. Patil,,Test case formation using UML activity diagram, Proceedings of " National Conference on Emerging Trends in Computer Technology (NCETCT-2012)" , World Journal of Science and Technology,India 2012.

[25] Biswal, Baikuntha Narayan, Test Case Generation and Optimization of Object-Oriented Software using UML Behavioral Models,2010,http://ethesis.nitrkl.ac.in/2923/

[26] B.B. Agarwal, S.P. Tayal, M. Gupta, Software Engineering and testing, Infinity Science Press, Jones and Bartlett, Hingham, Toronto, 2010.

[27] OMG, OMG Unified Modeling Language Specification version 1.4.2. 2001: OMG.