International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-8 Issue-6, August 2019

UML Activity Diagram Use for Functional Test Suit Generation and Redundancy Removal Supported Model Driven Testing

Runal G., Pramod Jadhav, Pruthviraj R. Pawar

***Abstract*: The method mixes up the extended finite state machine & UML activity diagram to generate the test model. H good coverage of test of all probable scenarios. Here an activity diagram describes the operation of the system, decision ere we have considered different coverage criteria for generating the test paths from the model for node transition from one action state to another. Also flow of control is represented. These will emphasis on sequence and condition of flow. It also gives idea about internal nodes.**

Refactoring is the process of altering an application's source code of its external behavior is not changing. The purpose of code refactoring is to improve some of the nonfunctional properties of the code, such as readability, complexity, maintainability and extensibility.

Refactoring can extend the life of source code, preventing it from becoming legacy code. The refactoring process makes future enhancements to such code a more pleasant experience. Refactoring is also known as reengineering.

Test cases tend to be massive in range as redundant take a look at cases square measure generated because of the presence of code smells, thus the requirement to scale back these smells.

Methods Statistical Analysis: This analysis adopts a proactive approach of reducing action at laws by police investigation the lazy category code smells supported the cohesion and dependency of the code and applying the inline category refactoring practices before take a look at case generation there by considerably avoiding redundant take a look at cases from being generated..

***Index Terms*: UML, sequence diagram, depth first search algorithm, software testing, test case generator, refactoring, redundancy test case.**

I. INTRODUCTION

Take a look at cases square measure assumed to mirror the first package underneath take a look at (SUT). Hence the effectiveness of action at law generated is associated to the standard of the ASCII text file of system underneath take a look at.

Revised Manuscript Received on August 30, 2019.

* Correspondence Author

Runal G.*, IT Department, Bharati Vidyapeeth Deemed to be University College of Engineering, Pune, India.

Prof. Pramod Jadhav, IT Department, Bharati Vidyapeeth Deemed to be University College of Engineering, Pune, India.

Prof. Pruthviraj R. Pawar, Computer Department, Bharati Vidyapeeth College of Engineering, Navi Mumbai, India.

Up the standard & liableness of take a look at cases generated improves the quality of take a look acting whereas an improvement in ASCII text file will enhance the standard of test cases generated.

SOA is a field of study manner that styles and develops package within the style of fast, low-cost, loosely coupled and simple integrated in heterogeneous environments. Project service square measure a concrete implementation of the SOA and that they are developed by mistreatment open standards like SOAP, WSDL, and UDDI supported XML. Since composite project services has been created for mission crucial services and sophisticated business processes, the composite project services testing is important to confirm the execution of whole business method is consistent and guarantee a better service quality and liableness. Code refactoring is the method of restructuring existing coding system dynamic the factorization while not dynamic its external behavior. Refactoring is meant to boost nonfunctional attributes of the software. Blessings embrace improved code readability and reduced quality.

II. SYSTEM IMPLIMENTATION

System Architecture

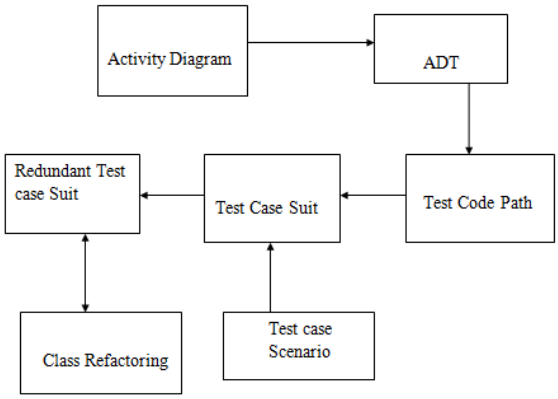


Figure 1: Architectural view of Functional Test case generation & Redundancy Check.

Methodology

To achieve all our objectives, we are going to use following methodology:

1. Input Activity Diagram:
2. ADT Generation
   - ✓ XML Code generation
3. Test Suit Generation

4. Redundant test case suite the system takes two inputs as activity diagram and generate test coverage path. Compare both the results and select most appropriate path for test case generation.This considers the various coverage criteria to generate test path that covers best test coverage of total situations. The presented work takes an activity diagram as input. Each activity diagram is well-known about making its ADT. This means to hold all needed details which are able to modify the model to look at capabilities and functionalities of all activity diagrams.
The outlines of each module are given as follows:

1) Generation of ADT:

Activity Dependency Table defines the loops, synchronization and strategies presenting the actions of the task are created technically utilizing every activity diagram. This decides to indicate the activities can move into the totally different entities.

2) Generation of ADG:

ADG are mechanically generated from the activity dependency table that is ADT. Names are given to every node by utilizing the symbols of the every task among the ADG. Here each node will display component or functionality within the activity diagram. As repetitive functionalities are specifies a consistent image among the ADT, only one node is made for them however what proportion times they're used among the activity diagram. It'll decreases the time needed for search operation within ADG.

3) Test cases Generation:

Here we are going to apply Depth First Search strategy for obtaining all the test paths consider for testing. The test path is designed from steps presenting the consecutive nodes. These steps will form complete path which start from first node to the end node in the ADG.

4) Redundancy controller:

One way to provide multiple "controllers" is to implement a "mirroring backup" so that another system also collects all data. If the system is controlling in real-time, having more than one CPU in the system is ideal, creating a bump less control system. This is a common requirement for redundancy.

5) Class Refactoring:

Much of refactoring is devoted to correctly composing methods. In most cases, excessively long methods are the root of all evil. The refactoring techniques in this group streamline methods, remove code duplication, and pave the way for future improvements. It shows how to safely move functionality between classes, create new classes, and hide implementation details from public access.

Algorithm Name: *GeneratingTestCasesSuite*
Input: All ADTs, ADGs of Activity Diagrams, and an empty table TC.
Output: Test cases suite Table *TC*
Steps:

```
Start
For each graph G in set of ADGs
    P[ ] = GetAllPaths(G) //P[ ]={P[1],P[2] ,...,P[n]}; where P[1]
    is the first path and P[n] is the last one in the G.
    Set j=0 //Counter for paths in P.
    For each path P[j]in G do
        Set i=0 //Counter for nodes in each path P[j].
        Add a new row in TC.
        For each node N[i]in P[j] do
            Add a new row in TC[j]
            Get the input and expected output of N[i] from the
            corresponding ADT.
            Put the input of N[i] under the column "Node Input"
            in TC[j][i].
            Put the expected output of N[i] under the column
            "Node Expected Output" in TC[j][i].
            i++
        End-For
        Put the input of N[0] under the column "Test case Input" in
        TC[j].
        Put the expected output of N[i-1] under the column "Test
        case Expected Output" in TC[j].
        j++
    End-For
End-For
Return TC
Stop
```

Algorithm 1 Detection of Lazy Class

I. Procedure Detection of Lazy Class
II. Get class name
III. If class is a super class, get sub class if class is new, add to class list
IV. If class already exists, add to existing class counter for each class or subclass c do
V. Read attributes, method and class call end for
VI. If attribute call in the class is new, add to counter ++i, else return i If method call in the class is new, add to counter ++j, else return j If class call in the class is new, add to counter ++k, else return k Repeat for all classes in source code SC
VII. For each class c do
VIII. Collate the classes, methods and attributes calling say x, y, z respectively end for
IX. For each LOC in class c do add to counter ++1
X. end for
XI. If ((i,j,k,l,x,y,z 3) return true; //(i.e lazy class detected) else return false;
XII. end procedure

Algorithm 2 Refactoring of Lazy Class

I. Procedure REFACTORING OF LAZY CLASS
II. Identify the movable class
III. If method in movable class not null
IV. Move method content to a host class of shortest distance if attribute not null, Push Field to the host class.
V. Remove all methods from movable class until method = null Remove all attributes from movable class until attributes = null Update method in the host class
VI. Update field in the host class
VII. Redirect reference calls from movable class to the host class Delete the lazy class
VIII. end procedure

III. RESULT ANALYSIS

Before Class Refactoring

Sr. No.	Class Name	Attributes	Methods
1	ATM	Location Branch	Show()
2	Card Scanner		AcceptCard(), ReadCard(), EjectCard(), ValidatePin()
3	Card Dispenser	Available Cash	SupplyCash(), GenerateRecepit()
4	ATM Card	pin, CardID, Acc	SetPIn(), GetPin(), GetAccount()
5	Bank Customer	Customer Name, Address, Email, Card, Acc	InsertCard(), SelectTransaction(), EnterPIn(), ChangePin(), WithdrawCash(), RequestTransactionSummary(), AcceptAmount()
6	Display Screen		Prompt(), AcceptInput()
7	Transaction	Date, Amount, Deposit	CalculateBalance(), SetTransaction(), GetAccountBalance(), CancelTransaction()
8	Account	Account Number, Balance, Trans	CalculateInterest(), UpdateAccount(), VerifyWithdrawAmount()
9	Saving Account	Interest Rate	CalculateInterest()
10	Current Account	Interest Rate	CalculateInterest()

Table 1 before Class Refactoring

Here, **Account** is Host class and movable classes are **SavingAccount** and **CurrentAccount**.

After Class Refactoring

Sr. No.	Class Name	Attributes	Methods
1	ATM	Location Branch	Show()
2	Card Scanner		AcceptCard(), ReadCard(), EjectCard(), ValidatePin()
3	Card Dispenser	Available Cash	SupplyCash(), GenerateRecepit()
4	ATM Card	pin, Card ID, Acc	SetPIn(), GetPin(), GetAccount()
5	Bank Customer	Customer Name, Address, Email, Card, Acc	InsertCard(), SelectTransaction(), EnterPIn(), ChangePin(), WithdrawCash(), RequestTransactionSummary(), AcceptAmount()
6	Display Screen		Prompt(), AcceptInput()
7	Transaction	Date, Amount, Deposit	CalculateBalance(), SetTransaction(), GetAccountBalance(), CancelTransaction()

8	Account	Account Number, Balance, Trans, Interest Rate	CalculateInterest(), UpdateAccount(), VerifyWithdrawAmount()

Table2 Result after Class Refactoring

We have update **Account Class** as this is host class of 'SavingAccount' and 'CurrentAccount'.

Analysis of Class

Before Class Refactoring

Sr. No.	Class Name	No. of Methods	Line of Code
1	Account	3	9
2	Saving Account	1	3
3	Current Account	1	3

Table 3 before Class Refactoring Analysis of Class

After Class Refactoring Analysis of Class

Sr. No.	Class Name	No. of Methods	Line of Code
1	Account	4	12

Table 4 after Class Refactoring Analysis of Class

Branch Coverage

Branch Coverage = $x = \frac{\text{No.of lines of code covered}}{\text{Total no.of Lines of code (20)}}$*100

Before Class Refactoring Branch Coverage

Sr. No.	Class Name	No. of Methods	Branch Coverage
1	Account	3	45%
2	Saving Account	1	15%
3	xCurrent Account	1	15%

Table 5 before Class Refactoring Branch Coverage

After Class Refactoring Branch Coverage

Sr. No.	Class Name	No. of Methods	Branch Coverage
1	Account	3	45%
2	Lazy Class (Merged both Class into host class)	1	15%

Table 6 after Class Refactoring Branch Coverage

Test Cases Generated

Before Class Refactoring TC generated

Sr. No.	Test Case	Scenario	Input	Expected Output	Actual Output
1	Saving Account -TC1	Calculate Interest	Interest Rate	Get Result with Float Value	Get Result with Float Value
2	Current Account-TC1	Calculate Interest	Interest Rate	Get Result with Float Value	Get Result with Float Value

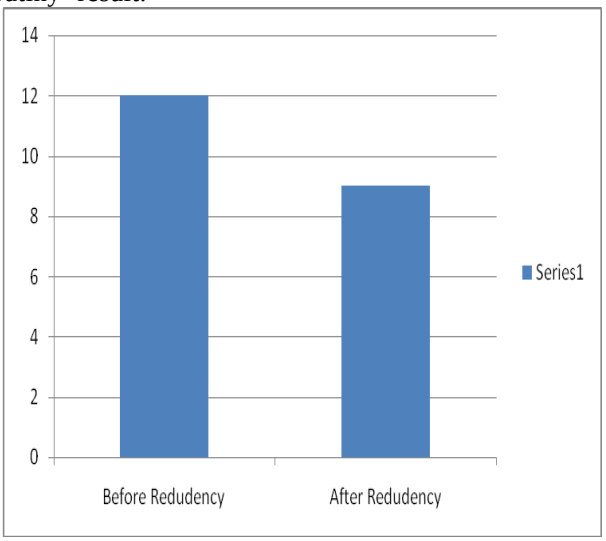
Table 7 before Class Refactoring TC generated

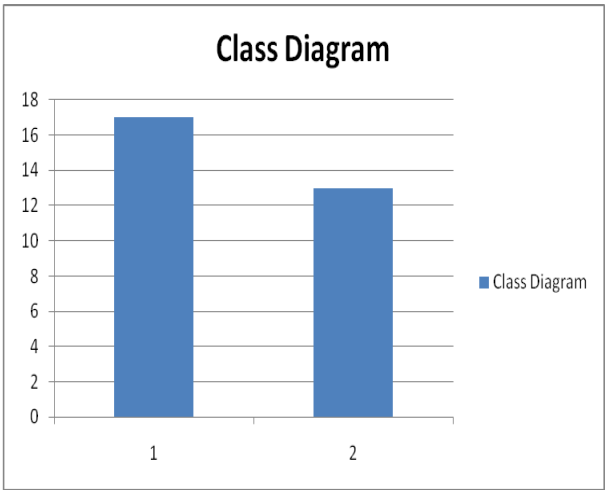
After Class Refactoring TC generated

Sr. No.	Test Case	Scenario	Input	Expected Output	Actual Output
1	Saving Account and Current Account -TC1*	Calculate Interest	Interest Rate	Get Result with Float Value	Get Result with Float Value

Table 8 after Class Refactoring TC generated

As shown within the Table one shows the analysis of lazy category refactoring supported cyclamate quality & branch coverage compared with the first allocate ASCII text file. An action at law decrease approach is incomplete if the quality of the action at law isn't ensured. One amongst the ways that to try and do this can be hard the branch coverage of the before and once refactoring code & scrutiny result.





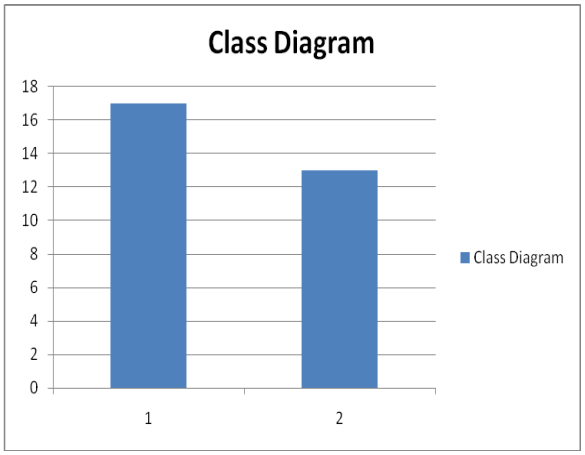


Fig 2. Refactoring results of no. of methods of classes for withdraw

Refactoring results of no. of methods of classes for pin changes

	Before Refactoring	After Refactoring
Class Diagram	17	13

Table 9 Refactoring results

IV. CONCLUSIION

According to our experience, the input activity diagram maintains the flexibility appropriately for defining the requirements. Also these diagrams are applicable for automatic process. The activity diagrams give some notation and these notations are referred for describing the model. The proposed method is able for creating more capable or proficient test suit by saving the time of tester. Also it saves the effort and increase quality of test cases which was generated by this method. In short, overall performance of testing process can be improved by this method. Here, we are presented several criteria of covering test path generation and algorithm for generating the test path automatically. During this paper, we have got shown the way to develop take a look at model and way to outline coverage criteria. Conjointly target to develop take a look at path generating algorithmic rule for Activity diagram given path. The end result shows that refactoring the ASCII text file before action at law generation has reduced the take a look at cases by thirty three percentage and inflated the branch coverage up to nine two percentage. Hence, the approach could be prospective action at law reduction technique. This means that the price and energy in testing is reduced by eliminating the code smells before action at law generation.

REFERENCES

1. Rajvir Singh, "Test Case Generation for Object-Oriented Systems: A Review" IEEE, Fourth International Conference on Communication Systems and Network Technologies, 2014.
2. Soma Sekhara Babu Lam et al. "Automated Generation of Independent Paths and Test Suite Optimization Using Artificial Bee Colony" Procedia Engineering, Elsevier pp. 191-200, 2012.
3. Utting, M., Pretschner, A., Legeard, B.: A taxonomy of model based testing approaches. Softw.Test.Verif. Reliab. **22**(5), 297–312 (2012).
4. Tripathy A, Mitra A. Test case generation using activity diagram and sequence diagram. In: Proceedings of International Conference on Advances in Computing. Springer; 2013. p. 121–9.
5. Lashari SA, Ibrahim R, Senan N. Fuzzy Soft Set based Classification for Mammogram Images. International Journal of Computer Information Systems and Industrial Management Applications. 2015; 7:66–73.
6. Ahmed M, Ibrahim R, Ibrahim N. An Adaptation Model for Android Application Testing with Refactoring. Growth. 2015; 9(10):65–74.
7. Vikas Panthi, Durga Prasad Mohapatra, "Automatic Test Case Generation using Sequence Diagram", International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868 Foundation of Computer Science FCS, New York, USA Volume 2– No.4, May 2012 – www.ijais.org.
8. Md Azaharuddin Ali et.al. "Test Case Generation using UML State Diagram and OCL Expression", International Journal of Computer Applications (0975 – 8887) Volume 95– No. 12, June 2014.
9. S. Shanmuga Priya et.al, " Test Path Generation Using UML Sequence Diagram", Volume 3, Issue 4, April 2013 ISSN: 2277 128X International Journal of Advanced Research in Computer Science and Software Engineering.
10. Ching-Seh Wu, Chi-Hsin Huang," The Web Services Composition Testing Based on Extended Finite State Machine and UML Model", 2013 Fifth International Conference on Service Science and Innovation.

AUTHORS PROFILE

Miss Runal G. M.Tech IT student in Bharati Vidyapeeth Deemed to be University College of Engineering, Pune. She was Completed her Graduation Degree 2016, from Avinashilingam University for Women's, Coimbotre.

Prof. Pramod Jadhav, he is working as an Assistant Professor in Bharati Vidyapeeth Deemed University College of Engineering, Pune. He has 12 Years' experience in Teaching. He is Pursuing PhD in Bharati Vidyapeeth Deemed Universirty, Pune.

Prof. Pruthviraj R. Pawar, he is working as an Assistant Professor in Bharati Vidyapeeth's College of Engineering, Navi Mumbai. He has 5 years' Experience in Teaching Field. He is Pursing PhD in Maharashi University of Information Technology, Lucknow.