A Novel Approach for Automated Test Path Generation using TABU Search Algorithm

A.V.K. Shanthi
Research Scholar, Sathyabama University,
Chennai, India

G. MohanKumar, PhD.
Prinicipal, Park Engineering College
Coimbatore,

ABSTRACT
Software testing is the last phase of the development cycle. The important role in software development is software Testing. In today's software industry, the design of software tests is mostly based on the tester's expertise, while test automation tools are limited to execution of preplanned tests only. Testing effort can be classified into three parts, they are test case generation, test execution and test evaluation. This paper presents a novel approach to generate the automated test paths. Due to the delay in the development of software, testing has to be done in a short time. This led to automation of testing because its efficiency and also requires less man power. In this proposed approach, by using one of the most standard Unified Modeling Language (UML) Activity Diagram, construct the Activity Dependency table(ADT), then generate the Test paths. Then the test path are prioritized by using the Tabu search algorithm. The prioritized test path can be used in system testing, regressing testing and integration testing. Then also form the Cyclomatic diagram to check the efficiency of the test scenario.

General Terms
Design, Development, Execution and Evaluation.

Keywords
Software Testing, Test Cases, UML (Unified Modeling Language), Activity Diagram, Tabu Search algorithm, Activity Dependency Table (ADT), Prioritization.

1. INTRODUCTION
To improve software productivity and quality, software engineers are increasingly applying data mining algorithms to various software engineering tasks. However mining software engineering data poses several challenges, requiring various algorithms to effectively mine sequences, graphs and text from such data. Software engineering data includes code bases, execution traces, historical code changes, mailing lists and bug data bases. They contains a wealth of information about a projects-status, progress and evolution. Using well established data mining techniques, practitioners and researchers can explore the potential of this valuable data in order to better manage their projects and do produce higher-quality software systems that are delivered on time and with in budget. Data mining can be used in gathering and extracting latent security requirements, extracting algorithms and business rules from code, mining legacy applications for requirements and business rules for new projects etc. When considering about the software testing, testing plays a important role. Testing effort can be classified into three parts, they are test case generation, test execution and test evaluation.

This is the most common style of automated testing.

- Create a test case.
- Run it and inspect the output
- If program fails, report bug and try later.
- If program passes, save the resulting outputs.
- In the future:

» Run the program and compare the output to the saved results.

» Report an exception when the current output and the saved output don't match.

Most commonly, we run these tests on the finished program, testing underlying functions by issuing commands through the program's GUI. Therefore, these are typically called GUI-level test tools.

But it leads to Various problems such as:

- Underestimated cost
- Your most technically skilled staff are tied up in automation

But especially,

Low power of regression testing

And,

Low attention to maintainability.

To improve maintainability in the face of a constantly changing user interface, split the design of the test cases from the automation of the features.

- . Describe the test cases as data that can be fed to the program
- . Describe the methods to set the value of each feature.

In this context, model-based testing appears as a promising approach to control software quality as well as to reduce the inherent costs of a test process, since test cases can be generated from the software specification, concomitantly to its development.

The problems concerning the testing techniques for feature, and particularly, feature interaction have been investigated. However, feature testing is as important as feature interaction testing, because the costs to find and correcting bugs during the feature interaction phase are higher, since feature interaction testing phase involves more than one feature.

This work focus on functional testing for banking applications (features). Test paths are generated from UML activity diagrams that represent a feature behavior. Due to the nature of this application (small size and sequence of actions and reactions), UML activity diagrams are presented as good artifacts. The idea is to reuse activity diagrams that are constructed by development teams to specify use cases with basic and alternative scenarios.

In this paper, we use UML activity diagrams as design specifications, and implemented the idea of automated test case generation for software as an application software

testing approach, to identify the most error prone paths in a software construct. Here we have proposed a heuristic technique for prioritization of test cases derived from the activity diagram using Activity Dependence Table and Tabu search algorithm.

We discuss the related work in the next section. Section 3 describes our proposed approach for automated test cases generation. Section 4 presents our implementation. Finally, we conclude the paper with discussions and future work.

2.RELATED WORK

In this section, we describes the representative researcher survey [3-5] of test cases generation based on UML activity diagrams. "Generating Test Cases from UML Activity Diagram based on Gray-Box Method," by Wang. L., Yuan, J., et al [3] defines a method for generates test cases from UML activity diagrams systematically, which modifies Depth First Algorithm (DFS) for automated generation. This paper does not fully handle fork-join structures. The deficiency of this paper is that any fork node has only two exit edges; evidently, these assumptions limit the applicable scope of the proposed algorithm. Another problem in this paper is that basic path defined, basic paths, can be found by the DFS algorithm, while the detailed walkthrough of the proposed algorithm shows that some test scenarios are not generated, especially when the test scenarios are derived from the fork-join parts of the activity diagrams .

"Using Anti-Ant-like Agents to Generate Test Threads from the UML Diagrams," by Li, H., Lam, C. P. [4] describes the concept of the thin-thread tree, the condition tree, and the data-object tree, as well their relationship with the UML activity diagrams. The previous works dealing with test scenario generations in activity diagrams did not consider data objects and input values. The proposed algorithms are incomplete. For example, for the example in Figure 1, it generates only two test scenarios. If we regard each activity as an atomic one, it should generate ten test scenarios.

"An Automated Approach to Generating Usage Scenarios from UML Activity Diagrams," by Chandler, R., Lam, C. P., Li, H., [5] defines the concept of UML activity diagram is a notation suitable for modeling a concurrent system in which multiple objects interact with each other. This paper proposes a method to generate test cases from UML activity diagrams that minimizes the number of test cases generated while deriving all practically useful test cases. In this method first builds an I/O explicit Activity Diagram from an ordinary UML activity diagram and then transforms it to a directed graph, from which test cases for the initial activity diagram are derived. This conversion is performed based on the single stimulus principle, which helps avoid the state explosion problem in test generation for a concurrent system.

Our related work is Automated Test case from UML Diagram – Class Diagram using Data Mining Approach.[11],[12] In this paper, we proposed a novel approach to generate test path from UML activity diagram using Tabu search algorithm.

3.TEST CASES GENERATION

In this section, we describe our test generation method. First, we generated the activity diagram from the software design. Write Parser in java to extract all possible information from the activity diagram. In Activity diagram each node represents an state. An activity diagram can contain any number of decision, fork, join and etc., for generating the test cases from the activity diagram we first extract the necessary information from the diagram. Based on the extracted information, an Activity Dependency Table(ADT) is generated. With the help of ADT test cases are generated, by applying the Tabu Search Algorithm, most prioritized test case are generated. Steps involved in the Generation of Test cases.

1. Draw the activity Diagram
2. Extract the necessary information
3. Generate the dependency table
4. From the dependency table generate the test case.
5. Applying Tabu Search algorithm, to generate the test path.
6. Constructing the cyclomatic diagram
7. Checking the test scenarios.

The Algorithm for generating test cases using Tabu search method is given below.

1 Draw the Activity diagram.
2 From the Activity diagram generate the dependency table.
3 Repeat the steps for all states.
  a. Store the start state in the array.
  b. By using the dependency the table get the next state.
  c. If next state is the decision state then,
    (i) Copy the entire path into the next empty location and store the value.
    (ii) Continue in the true side until it reaches end state.
    (iii) For the false side just add the end state and continue.
  d. If the next state is the end state,
    (i) Stop the process.
    (ii) Check any other path is incomplete, if yes then continue with there.
    (iii) Else exit.
  e. If the next state is the ordinary state, then add the state at the end of the current path.
4 For assigning weightage, use dependency table.
  a. Start the process with the start state assign the value as one.
  b. Using the dependency table check the next state.
  c. Increment the value and assign it to the next.
  d. If the current state is decision, then
    (i) Increment the value and assign it to the true side of the decision.
    (ii) And also for the false side of the decision.
  e. Repeat the step until it reaches the end state.
5 To calculate fitness value,
  a. for each node calculate the number of incoming node(a) and the number of outgoing nodes(b).
  b. by multiplying a and b, f=a*b , we get fitness value.
6 For Appling Tabu search algorithm, create three variable called tabuin ,tabuout and best.
  7 Calculate the initial test path value , store it in the best and store the path in tabuin.
  8 Calculate the neighborhood test path value.
  9 Compare the current value with the best.

a. if best is greater than the current value,
   (i) store the current path in tabuout
   (ii) then goto step no 10
b. if current value is greater than the best,
   (i) store the current value in the best
   (ii)add the current path to the tabuin

10 Repeat step 8 until all path have been covered.

11By using the tabuin we can get the prioritized test path.

12Exit.

4. CASE STUDY

The proposed method is evaluated by the Activity diagram (Figure 1) of Banking System – Cash Withdrawal or Cash Deposit process, created by using rational rose software

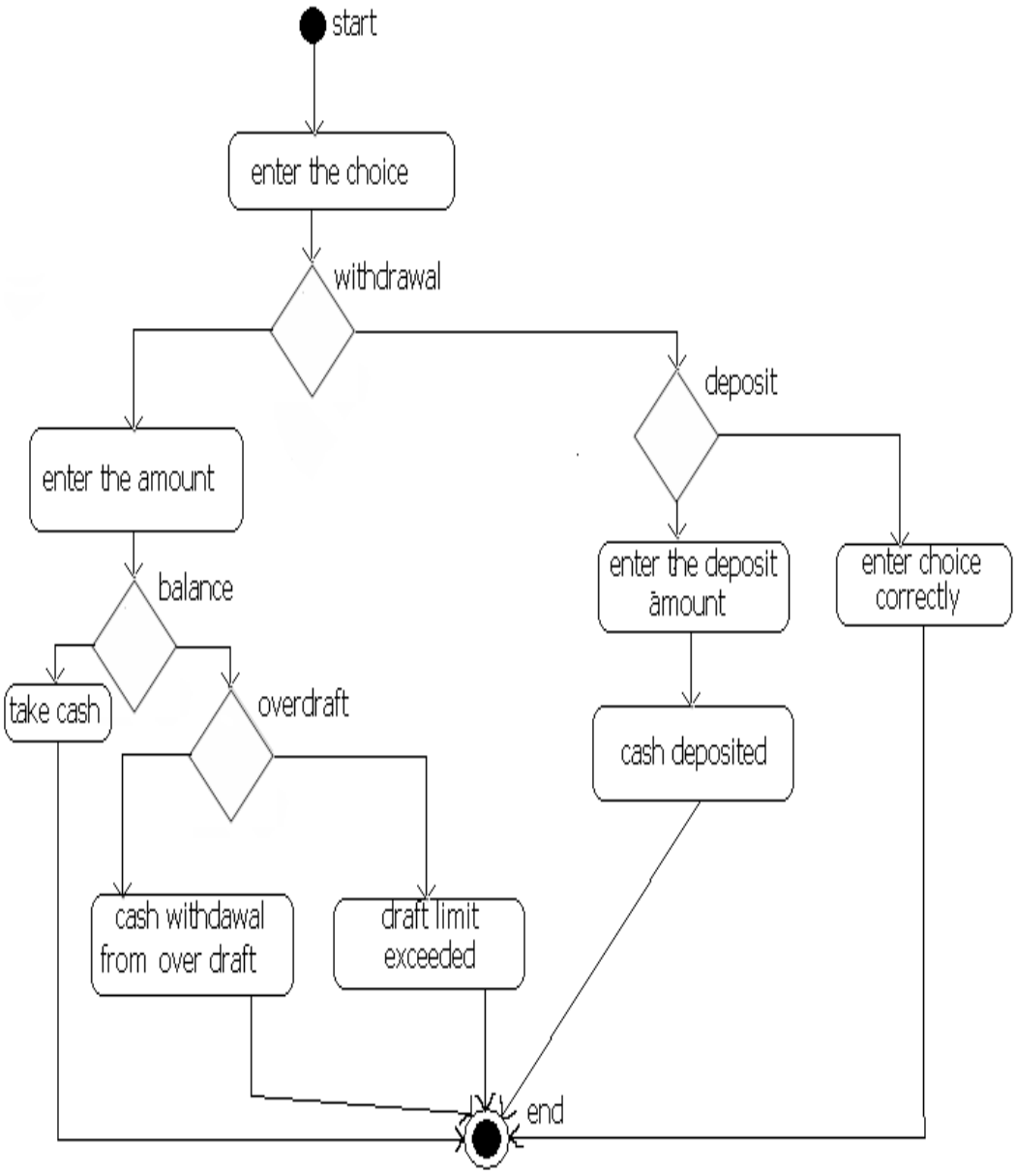

Figure 1 : Activity Diagram

Table 1: Activity Dependency Table(ADT)

Symbol	Activity Name	Dependency	Input	Expected output
A	Start			
B	Enter the Choice	A	Choice	Type of choice
C	Check withdrawal	B	Choice	True(withdrawal) False(check deposit)
D	Check deposit	B	Choice	True(deposit) false(enter choice correctly)
E	Enter the amount	C	Valid amount	Check(balance)
F	Check balance	E	Amount	True(take cash) False(check over draft)
G	Check over draft	F	Amount	True(take cash) False(limit exceeded)
H	Enter the deposit amount	D	Valid amount	Cash deposited
I	Cash deposited	H	Cash	End
J	Take cash	F G	Cash	End

From the collected information, the Activity dependency table(Table.1) is generated. The table consists of the following fields:- symbol, Activity Name , Dependency, Input and Expected output, where Symbol is used to indicate the state or the decision present in the diagram, Dependency indicate the previous node of the activity. Input and Expected output is also included in the Table 1.

Next, from the Activity Dependency Table the possible test paths are generated. The test paths will began with start state and terminates at the end state. From the start state move along the ADT till it reaches the end state with the help of dependency. The part of generating the test paths are described in the above algorithm step 3. Then the valid test paths are generated (Table 2).

Table 2: Test Path Obtained

Start → enter the choice → withdrawal → enter the amount → balance → take cash → end
Start → enter the choice → withdrawal → enter the amount → balance → over draft → cash withdrawal from over draft → end
Start → enter the choice → withdrawal → enter the amount → balance →over draft → draft limit exceed → end
Start → enter the choice → withdrawal → deposit → enter the amount for deposit → cash deposited → end
Start → enter the choice → withdrawal → deposit →enter the choice correctly → end

Once the test paths are obtained, apply the Tabu search Algorithm, the test cases can be prioritized.. The method for obtaining the most prioritized test path is described in the above algorithm (step 4 to step11). Finally, the most prioritized test path is obtained.(Table 3)

Table 3: The Most Prioritized Path

Start → enter the choice → withdrawal → enter the amount → balance →over draft → draft limit exceed → end

Table 4: Test Path Table

Test Path Number	Test Path Node	Node Input	Node Expected Output	Test Path Input	Test Path Output
1	Start		enter choice	valid amount and correct balance	take cash
	enter the choice	choice	check for withdrawal		
	check withdrawal	choice	true(withdrawal)		
	enter the amount	valid amount	check for balance		
	check balance	amount	true(balance)		
	take cash	amount	end		
	End				
2	Start		enter choice	deposit amount	deposited
	enter the choice	choice	check for withdrawal		
	check withdrawal	choice	false(withdrawal)		
	check deposit	choice	true(deposit)		
	enter the deposit amount	valid amount	insert into the account		
	End				
3	Start		enter choice	invalid choice	Warning the user
	enter the choice	choice	check for withdrawal		
	check withdrawal	choice	false(withdrawal)		
	check deposit	choice	false(deposit)		
	enter the correct choice	choice			
	End				
4	Start		enter choice	valid amount ,low balance and high overdraft	take cash
	enter the choice	choice	check for withdrawal		
	check withdrawal	choice	true(withdrawal)		
	enter the amount	valid amount	check for balance		
	check balance	amount	false(balance)		
	check over draft	amount	true(over draft)		
	take cash	amount	end		
	End				
5	Start		enter choice	valid amount ,low balance and no overdraft	no balance
	enter the choice	choice	check for withdrawal		
	check withdrawal	choice	true(withdrawal)		
	enter the amount	valid amount	check for balance		
	check balance	amount	false(balance)		
	check over draft	amount	false(over draft)		
	End				

The obtained test paths are evaluated by checking with expected inputs and outputs (Table 4). Finally, The test paths are validated by means of Cyclomatic diagram. Generating the Cyclomatic diagram from the Activity Dependency Table. From it, calculate the cyclomatic complexity of the proposed test paths.

$$V(G) = E – N + 2$$

where V(G) is the cyclomatic complexity E is the number of edge, N is the number of node present in the diagram.

The test paths are validated using cyclomatic complexity value. From the cyclomatic diagram (figure 2), the Cylomatic complexity value is 5. and the number of test paths generated by the proposed algorithm is also 5 .

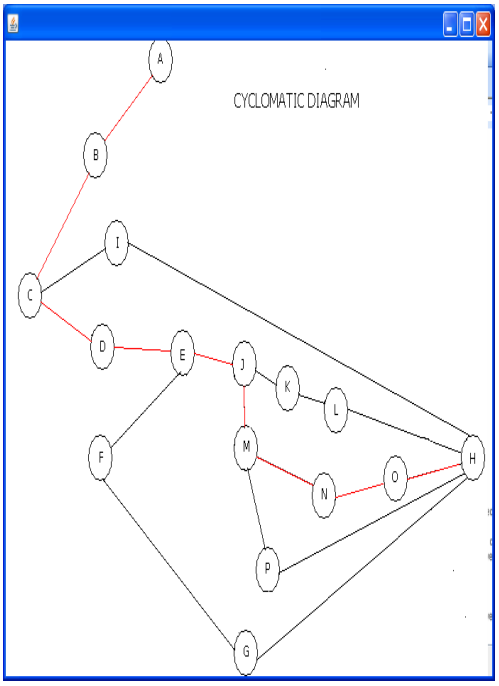

Figure 2. Cyclomatic diagram

5. DISCUSSION AND FUTURE WORK

In this paper we presented the test paths generation by means of UML Activity diagram using Tabu search Algorithm which best test paths are optimized and the test paths are validated by prioritization**.** Our approach is significant due to the following reasons. First, our approach is capable to detect errors like errors in loop, synchronization faults easier than the existing approaches. Second, test case generated in our approach may help to identify location of a fault in the implementation, thus reducing testing effort. Third, our heuristic method for test path generation inspires the developers to improve the design quality, find faults in the implementation early, and reduce software development time. Fourth, following our approach, it is possible to build an automatic tool easily. This automatic tool will reduce the cost of software development and improve quality of the software

6. ACKNOWLEDGEMENT

The authors gratefully acknowledge to Mr. V.Rajeshkanna , IV M.Sc., Software Engineering, Sathyabama University for the valuable help in developing the code in java**.**

7. REFERENCES

[1] M.Prasanna, S.N.Sivanandam, Venkatesan, R.Sundarrajan, 15, 2005,"A SURVEY ON AUTOMATIC TEST CASE GENERATION", Academic Open Internet Journal.

[2] M.Prasanna, K.R. Chandran, ” Automatic Test Case Generation for UML Object diagrams using Genetic Algorithm”, Int. J. Advance. Soft Comput. Appl., Vol. 1, No. 1, July 2009.

[3] Wang. L., Yuan, J., Yu, X., , Hu, J., Li , X., Zheng G., “Generating Test Cases from UML Activity Diagram based on Gray-Box Method,” National Natural Science Foundation of China, 2005.

[4] Li, H., Lam, C. P., “Using Anti-Ant-like Agents to Generate Test Threads from the UML Diagrams,” TestCom 2005, LNCS 3502, pp. 69 – 80, 2005.

[5] Chandler, R., Lam, C. P., Li, H., “An Automated Approach to Generating Usage Scenarios from UML Activity Diagrams,” Proceedings of the 12th Asia-Pacific Software Engineering Conference, 2005.

[6] Chen, M., Qiu, X., Li, X., “Automatic Test Case Generation for UML Activity Diagrams,” National Natural Science Foundation of China, AST’06, 2006.

[7] Xu, D., Li, H., Lam, C.P., “Using Adaptive Agents to Automatically Generate Test Scenarios from the UML Activity Diagrams,” Proceedings of the 12th Asia-Pacific Software Engineering Conference, 2005.

[8] Andriole, S. J., Software Validation, Verification, Testing and Documentation, Petrocelli Books, 1986.

[9] Kang, S., Shin, J., Kim, M., “Interoperability test suite derivation for communication protocols,” Computer Networks Journal, Vol. 32, No. 3, 2000.

[10] Gao, J., Taso, H. S. J., Wu, Y., Testing and Quality Assurance for Component-based Software, Artech House Inc., 2003.

[11] A.V.K. Shanthi, Dr.G.Mohan Kumar, “Automated Test Case From UML Diagram Using Data Mining Approach”, CiiT International Journal of Software Engineering and Technology, Vol3.No3, March 2011.

[12] A.V.K. Shanthi, Dr.G.Mohan Kumar, “Automated Test Cases Generation For Object Oriented Software”, Indian Journal of Computer Science and Engineering, Vol:2, issue 4,Sep2011.

[13] Baikuntha Narayan Biswal, Pragyan Nanda, Durga Prasad Mohapatra, 2008 IEEE, "A Novel Approach for Scenario-Based Test Case Generation", International Conference on Information Technology.

[14] Chang-ai Sun, 2008 IEEE, "Transformation-based Approach to Generating Scenario-oriented Test Cases from UML Activity Diagrams for Concurrent Applications", Annual IEEE International Computer Software and Applications Conference.

[15] Bin Lei, Linzhang Wang, "Xuandong Li, UML Activity Diagram Based Testing of Java Concurrent Programs for Data Race and Inconsistency ", 2008 International Conference on Software Testing, Verification, and Validation.

[16] P. Samuel, R. Mall, A.K. Bothra,2008 "Automatic test case generation using unified modeling language (UML) state diagrams ",Published in IET Software.

[17] Emanuela G. Cartaxo, Francisco G. O. Neto and Patr´ıcia D. L. Machado, "Test Case Generation by means of UML Sequence Diagrams and Labeled Transition Systems", IEEE 2007.

[18] Hyungchoul Kim, Sungwon Kang, Jongmoon Baik, Inyoung Ko, "Test Cases Generation from UML Activity Diagrams ", Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing.

[19] Supaporn Kansomkeat and Sanchai Rivepiboon, "Automated-Generating Test Case Using UML Statechart Diagrams ",SAICSIT 2003.

[20] Santosh Kumar Swain, Durga Prasad Mohapatra, and Rajib Mall, "Test Case Generation Based on Use case and Sequence Diagram", Int.J. of Software Engineering, IJSE Vol.3 No.2 July 2010.