TEST CASE GENERATOR BASED ON UML ACTIVITY DIAGRAM

LOH KHA BEE

This report is submitted in partial fulfillment of requirement for the

Bachelor Degree of Computer Science (Software Engineering)

Faculty of Computer System and Software Engineering
Universiti Malaysia Pahang

December 2014

ABSTRACT

Software testing serves to identify and report bugs exist in system. Test case is required to aid the process of testing by giving the event that can and should be verified. Nowadays, there are testing tool that required tester to enter system execution script. This can affect the quality of test due to lack of experience of tester or human mistakes. In this project, test case generator based on Unified Modeling Language (UML) Activity Diagram is developed. With generating of flow graph from UML Activity Diagram, test cases are able to be output. Therefore, the test case number of particular UML Activity Diagram can be determined.

ABSTRAK

Pengujian perisian berfungsi untuk mengenal pasti dan melaporkan pepijat yang wujud dalam sistem. Kes ujian diperlukan untuk membantu dalam proses ujian dengan memberi acara yang boleh dan harus disahkan. Kini, terdapat alat ujian yang memerlukan penguji skrip untuk memasukkan skrip pelaksanaan sistem. Hal ini boleh menjejaskan kualiti ujian disebabkan kekurangan pengalaman penguji atau kesilapan manusia. Dalam projek ini, penjana kes ujian berdasarkan UML Activity Diagram dibina. Dengan menjana graf aliran dari UML Activity Diagram, kes-kes ujian dapat dijana. Oleh itu, bilangan kes ujian dalam UML Activity Diagram tertentu boleh dikenalpasti.

TABLE OF CONTENTS

BORANG PENGESAHAN STATUS TESIS ii

STUDENT DECLARATION iv

SUPERVISOR DECLARATION v

ACKNOWLEDGEMENT vi

ABSTRACT vii

ABSTRAK viii

TABLE OF CONTENTS ix

LIST OF TABLES xii

LIST OF FIGURES xiii

LIST OF ABBREVIATION xiv

LIST OF APPENDICES xv

CHAPTER 1 1

INTRODUCTION 1

1.1 PROBLEM STATEMENT 1

1.2 OBJECTIVE 2

1.3 SCOPE 2

1.4 THESIS ORGANIZATION 2

CHAPTER 2 3

LITERATURE REVIEW 3

2.1 UML ACTIVITY DIAGRAM 3

2.2 TEST CASE GENERATOR 5

2.3 CYCLOMATIC COMPLEXITY 5

2.4 FIVE TECHNIQUES ON GENERATING TEST CASE BASED ON UML ACTIVITY DIAGRAM 7

2.4.1 Test case generation with input output explicit activity diagram (IOAD) 7

2.4.2 Test case generation with Condition-Classification Tree Method 7

2.4.3 Test case generation with sub activity diagram 8

2.4.4 Test case generation for user acceptance testing 8

2.4.5 Test case generation with enhanced technique 8
2.5 IMPLEMENTATION OF THE FIVE TECHNIQUES 8
2.5.1 IOAD technique on generating test cases 9
2.5.2 Condition-Classification Tree Method on generating test cases 10
2.5.3 Sub activity diagram on generating test cases 11
2.5.4 User acceptance testing on generating test cases 12
2.5.5 Enhanced technique on generating test cases 13
2.6 COMPARISON OF THE FIVE TECHNIQUES 16
2.7 PROPOSED TECHNIQUE 18
CHAPTER 3 20
METHODOLOGY 20
3.1 METHODOLOGY 20
3.2 HARDWARE AND SOFTWARE 25
3.3 GANTT CHART 28
CHAPTER 4 29
DESIGN AND IMPLEMENTATION 29
4.1 FRAMEWORK AND MODEL 29
4.1.1 UML Activity Diagram Converter Algorithm 30
4.1.2 Test Case Generator Algorithm 33
4.1.3 The flow chart of TCGenerator 36
CHAPTER 5 37
RESULTS AND DISCUSSION 37
5.1 PROPOSED MODEL ACTUAL OUTPUT 37
5.2 PROPOSED MODEL ALGORITHM EXPLANATION 38
5.3 COMPARISON OF PROPOSED MODEL WITH REVIEWED TECHNIQUES 39
5.4 RESULT ANALYSIS 41
5.4.1 Test hypotheses 42
5.5 REAL LIFE IMPLEMENTATION OF TCGENERATOR 43

5.6 RESEARCH CONSTRAINTS 53
5.7 SUGGESTION AND ENHANCEMENT OF PROJECT 54
5.7.1 Contribution of project 54
5.7.2 Enhancement of project 55
CHAPTER 6 56
CONCLUSION 56
6.1 CONCLUSION OF RESEARCH 56
6.2 OBJECTIVES ACHIEVEMENT 56
6.3 METHODOLOGY AND RESEARCH IMPLEMENTATION 56
6.4 FUTURE SUGGESTION AND ENHANCEMENT 57
APPENDIX A 58
Gantt Chart 58
APPENDIX B 60
Turnitin Report 60
APPENDIX C 61
TCGenerator Source Code 61
APPENDIX D 64
User Manual 64
REFERENCES 66

LIST OF TABLES

Table 1 : Symbols of UML Activity Diagram 4

Table 2 : ADT of Case Study 13

Table 3 : Comparison of Techniques 16

Table 4 : Hardware Description 25

Table 5 : Software Description 26

Table 6 : Activity and Respective Numeric Representation 31

Table 7: Comparison of Proposed Model with Reviewed Techniques 39

Table 8 : Activity and Respective Numeric Representation of ATM 45

Table 9 : Test Case of UML Activity Diagram of ATM (Withdraw Money) 48

LIST OF FIGURES

Figure 1 : Control Flow Subgraphs 6

Figure 2 : Steps in CCTM 7

Figure 3 : Example Activity Diagram for Case Study 9

Figure 4 : Activity Diagram Based on IOAD 10

Figure 5 : Condition-Classification Tree Tables of Case Study 11

Figure 6 : Sub Activity Diagram for "Check availability" Activity in Case Study 11

Figure 7 : IFD of Case Study 12

Figure 8 : ADG before Reduction of the ADT Table 2 14

Figure 9 : ADG after Reduction of the ADT Table 2 15

Figure 10 : Flow Chart of Research Framework 21

Figure 11 : Framework of Proposed Test Case Generator 29

Figure 12 : Illustration of Flow Graph Generated 32

Figure 13 : Flow Chart of TCGenerator 36

Figure 14 : Output of TCGenerator 37

Figure 15 : Adjacency Matrix 38

Figure 16 : ArrayList 38

Figure 17 : Increment of Activity Depth 42

Figure 18 : j Indicates Depth Value 43

Figure 19 : UML Activity Diagram of ATM (Withdraw Money) 44

Figure 20 : CFG of UML Activity Diagram of ATM (Withdraw Money) 46

Figure 21 : Output of UML Activity Diagram of ATM (Withdraw Money) 47

LIST OF ABBREVIATION

ABBREVIATION	MEANING
ADT	Activity Dependency Table
ADG	Activity Dependency Graph
ATM	Automatic Teller Machine
CCTM	Condition-Classification Tree Method
CFG	Control Flow Graph
DFS	Depth First Search
GUI	Graphical User Interface
IFD	Interaction Flow Diagram
IFG	Interaction Flow Graph
IOAD	Input Output Explicit Activity Diagram
ISO	International Organization for Standardization
IT	Information Technology
MGS	Modified Graph Search
PIN	Personal Identification Number
UML	Unified Modeling Language

LIST OF APPENDICES

APPENDIX A : Gantt Chart....................58

APPENDIX B : Turnitin Report....................60

APPENDIX C : TCGenerator Source Code....................61

APPENDIX D : User Manual....................64

CHAPTER 1

INTRODUCTION

Software is written by fallible human. Therefore, existing of defects in .the software products is inevitable. These defects can cause risks of failure when system execution [8]. However, the level of these risks of failure can be reduced through testing to uncover the bugs or any anomalies in the system. Then, the identified bugs and anomalies can be removed through debugging. For testing to be carried out, test cases are required to determine the test condition to be covered during the test. Tester needs to design the test case by own and this may affect the quality of test result due to lack of experience or human mistakes. This thesis covers research on methodology to generate test cases based on UML Activity Diagram.

1.1 PROBLEM STATEMENT

Testing is needed to be performed by tester to verify and validate the input data fulfil the requirement of a software functionality. Therefore, test case is needed. The demand for automated test has grown drastically over the years. Current tool on automate the test execution has grown in line with the existence of automated tool. Oracle OATS, HP, and IBM Rational Test Workbench are among the automated tools used by software industry. However, these automated tools are not able to generate test cases based on software specification specified. The tools are made to provide tester to enter system execution script. Thus, quality of test result is highly affected by the quality of script produced by the tester due to lack of experience or human mistakes. Another, poor verification process occurs when the test case result is traceable to the system output that has been prepared before.

Although lots of diagram has been created such as UML Activity Diagram during software analysis and design, tester still designs the test case based on the software requirement specification. By generating test case based on the UML Activity Diagram can bridge the gap between the requirement specification and test case design.

1.2 OBJECTIVE

The objectives of the research are to:

i. Convert UML Activity Diagram into a McCabe flow graph.

ii. Modify Depth First Search (DFS) algorithm to generate test cases based on McCabe flow graph.

iii. Develop a prototype that can automatically generate the test cases based on a UML Activity Diagram without user interference of the process.

1.3 SCOPE

The scopes of the thesis are:

i. Research on test case generator based on Unified Modeling Language (UML) Activity Diagram. Test basis is limited to activity diagram.

ii. Produce the test case in a form of flow of execution which similar to test script but the generator will not produce the test data.

1.4 THESIS ORGANIZATION

This thesis consists of six chapters. Chapter 1 will discuss on introduction to research. Chapter 2 will discuss on the literature review of existing research or system and the technique/ method/ hardware/ technology currently exists. Chapter 3 will discuss on the overall approach and framework of research. Chapter 4 will cover the framework and model development through flow work. Chapter 5 will give the results and discussion of the research. Lastly, Chapter 6 will conclude the entire research work.

CHAPTER 2

LITERATURE REVIEW

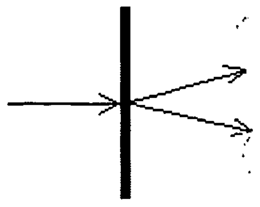
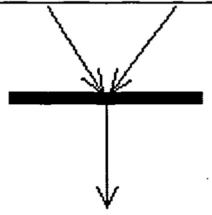
This chapter gives explanation regarding to generating test cases based on UML Activity Diagram. There are seven sections in this chapter. Section 2.1 describes about the UML activity diagram, section 2.2 describes about the test case generator, section 2.3 explains about what is Cyclomatic Complexity, section 2.4 gives five techniques on generating test cases based on UML Activity Diagram, section 2.5 shows the implementation of the techniques with case study, section 2.6 gives the comparison of the techniques, and section 2.7 is the proposed technique on test case generator.

2.1 UML ACTIVITY DIAGRAM

UML are commonly used as the design blueprints. It was created in the year 1990 by the three famous amigos named Grady Booch, Ivar Jacobson, and James Rumbaugh. In year 2000, UML was approved by the International Organization for Standardization (ISO) as a standard modelling language. Now, it is used by the industry as a standard language in modelling object-oriented software system. Since UML has the capability in modelling requirements, it becomes the important sources for test case generation. One of the diagrams is the UML Activity Diagram. Unlike other diagrams in the UML, activity diagram does not show clear origins in the previous work of the three famous amigos. It combines the elements from several techniques such as the event diagrams of Jim Odell, workflow modelling, and also the Petri nets. UML Activity diagram is suitable to be used to describe system behaviour since it has the capability to capture business process, workflow, and interaction scenarios [5]. Another, the activity dependencies are clearly depicted through the activity diagrams. Therefore, it is worth to study the generation of test case from the activity diagram. The following Table 1 gives the symbols used to model the system behaviour by activity diagram.

Table 1 : Symbols of UML Activity Diagram

Symbol	**Name**	**Function**
●	Start state	Indicates the beginning of the activities.
◉	End state	Indicates the end of the flow of activities.
◇	Decision, Merge	For decision, one transition line will connect to the diamond shape and with multiple transition lines coming out from the diamond shape. For merge, multiple transition lines will connect to the diamond shape and then only a single transition line coming out from it. The transition line will be labelled with guard condition indicates with "[ ]".
⬭	Activity	Show the action in the diagram.
□	Swim lane	A vertical column that used to model the activity's procedural flow of control between the objects that execute the action. The object can be person, organizations, or any responsible entities.
→	Transition	An arrow that indicates the flow from one action to another action.

	Fork	The fork represents two action sequences that are done in parallel.
	Join	The join represents two actions are rejoin back to a single action sequence.

2.2 TEST CASE GENERATOR

Test case generator is in high demand especially in this rapid Information Technology (IT) developing era. The complexity and size of software systems keep on expanding and this has caused the manual testing becomes error-prone. Therefore, the developing of automatic test case generator is believed to make the process of testing becomes more efficient as well as to reduce the numbers of errors and faults [7]. Model-based approaches are usually used in test case generation [13]. There are many existing test case generator developed with UML diagrams as the source to obtain software system requirements specification. Among them is the activity diagram where system behaviour is clearly shown. This is the key that determines the efficiency of the generator in producing the test cases as the quality of test cases depend on how much functionalities of system under test can be covered. A test case generator should fulfil the functionality of producing test case with full test coverage. In this project, flow graph which represents the UML Activity Diagram will be used as input to generate the test cases.

2.3 CYCLOMATIC COMPLEXITY

Cyclomatic Complexity serves as the software metric that measures the number of basic path in a program. Cyclomatic Complexity is developed by Thomas J. McCabe, Sr. in 1976 [2]. The software metric can be used to validate the number of test case generated. By defining the number of independent path in a system, the level of complexity of a program can be indicated. Cyclomatic complexity is implemented through the control flow graph which is a type of graph that uses the graph notations such as directed arrow that connects two vertices to show the paths travelled by the system during execution. The vertex is represented with circle labelled with activity. By converting an activity diagram into the control flow graph, the complexity of a program can be calculated using the formula.

Flow graph consist of vertices and edges [2]. Each vertex is connected by the edge. Basic flow of the flow graph [9] is listed in Figure 1.

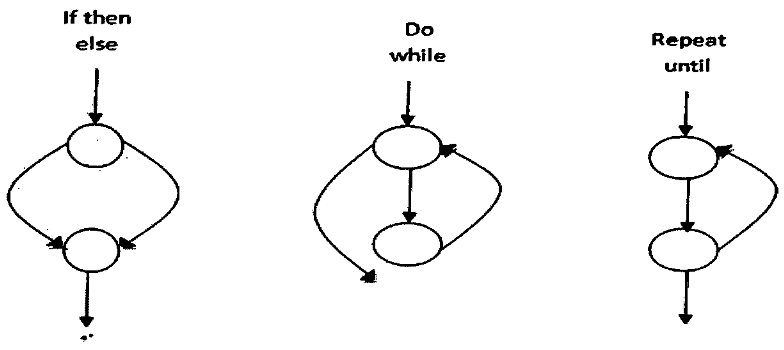


Figure 1 : Control Flow Subgraphs

The complexity M is defined as following:

M = E – N + 2P

Where E = Number of edges

N = Number of vertices

P = Number of exit vertices

For graph with the condition where the exit point is directed back to the entry point [1] or graph with closed region, the complexity M is defined as following:

M = R + 1

Where R = Number of closed region

McCabe proposed the Basis Path Testing which tests the linearly independent path of the program [2]. The testing technique would produce the same result as the Cyclomatic Complexity.

2.4 FIVE TECHNIQUES ON GENERATING TEST CASE BASED ON UML ACTIVITY DIAGRAM

The following gives the description of five test case generation techniques.

2.4.1 Test case generation with input output explicit activity diagram (IOAD)

The test case generation with IOAD emphasizes only on the external interaction of the system. The internal processing of the system is not taking in consideration [8]. The test paths are constructed based on the input of data by user and also the feedback produced to user. The IOAD method of test case generation can minimize the number of test cases. For implementation, I/O explicit activity diagram is first constructed based on the main activity diagram. Then, it is transformed into a directed graph. The test case derived is applying the single stimulus principle to avoid state explosion problem for a concurrent system [12].

2.4.2 Test case generation with Condition-Classification Tree Method

Condition-Classification Tree Method (CCTM) is the extended version of the Classification-Tree Method (CTM) [11]. This method uses the conditional branches to mark as to which test case covers which of the branches of the conditions in the activity diagram [8]. Figure 2 illustrates the three steps involved in CCTM.

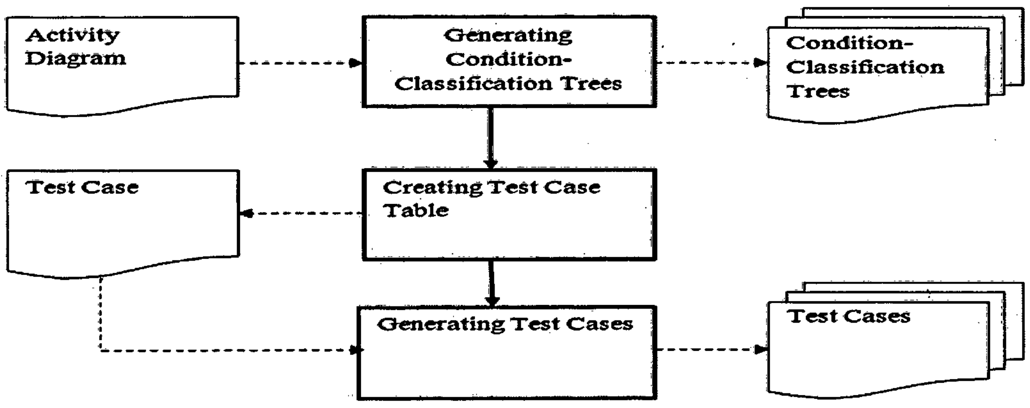


Figure 2 : Steps in CCTM

In step of generating condition-classification trees, the decision points and their respective guard conditions are extracted to be used in constructing the condition-classification tree. In step of creating test case table, the number of guard conditions related to the decision point in consideration is counted. The number is used to determine the arrangement of the decision points from left to right in ascending order. After arranging all the decision points, a table in grid form is drawn under each decision points where the column is correspond to the guard conditions and the row is correspond to the potential test cases [11]. In step of generating test cases, option vertex that has relationship with the vertices from correspond left trees is marked together on the same row. The operation of marking starts from left to right [11].

2.4.3 Test case generation with sub activity diagram

The sub activity diagram method gives a particular activity a more in depth view. The selected activity is further expanded into more details and is presented as separate activity diagram from the main activity diagram where it is derived. Then, a round-robin strategy is used to integrate all the test case at different level activity diagram into a test case for the whole system [12].

2.4.4 Test case generation for user acceptance testing

For user acceptance test, real user group are chosen to test the system. For the user to show interest and understand what they need to do with the system to be tested, the activity diagram is converted into Interaction Flow Diagram (IFD) [8]. The IFD shows the input or output related to the actions and also the role that performs the action. The activity performed by the system itself is excluded from the IFD. Then, Interaction Flow Graph (IFG) is derived based on the IFD. Each loop is traversed once and Depth First Search (DFS) is used to generate all valid paths for test case.

2.4.5 Test case generation with enhanced technique

With the enhanced technique, the complexity of activity diagram is reduced by first converting the details in the diagram into an Activity Dependency Table (ADT) [6]. Then, the Activity Dependency Graph (ADG) is derived from the ADT by including all the activity. Test path is then generated from the graph by using path coverage criterion with DFS. Then, activities in loop are grouped together in order to minimize the test paths [8].

2.5 IMPLEMENTATION OF THE FIVE TECHNIQUES

In this section, the case study by Khurana et al [8] is used to show the implementation of each technique. It is an activity diagram that shows the activity flow of a shipping company. The process cover form firstly customer placing their orders till the last step of receiving is generated. Through the example, explanations are given on the workflow of each techniques and how the test cases are generated.

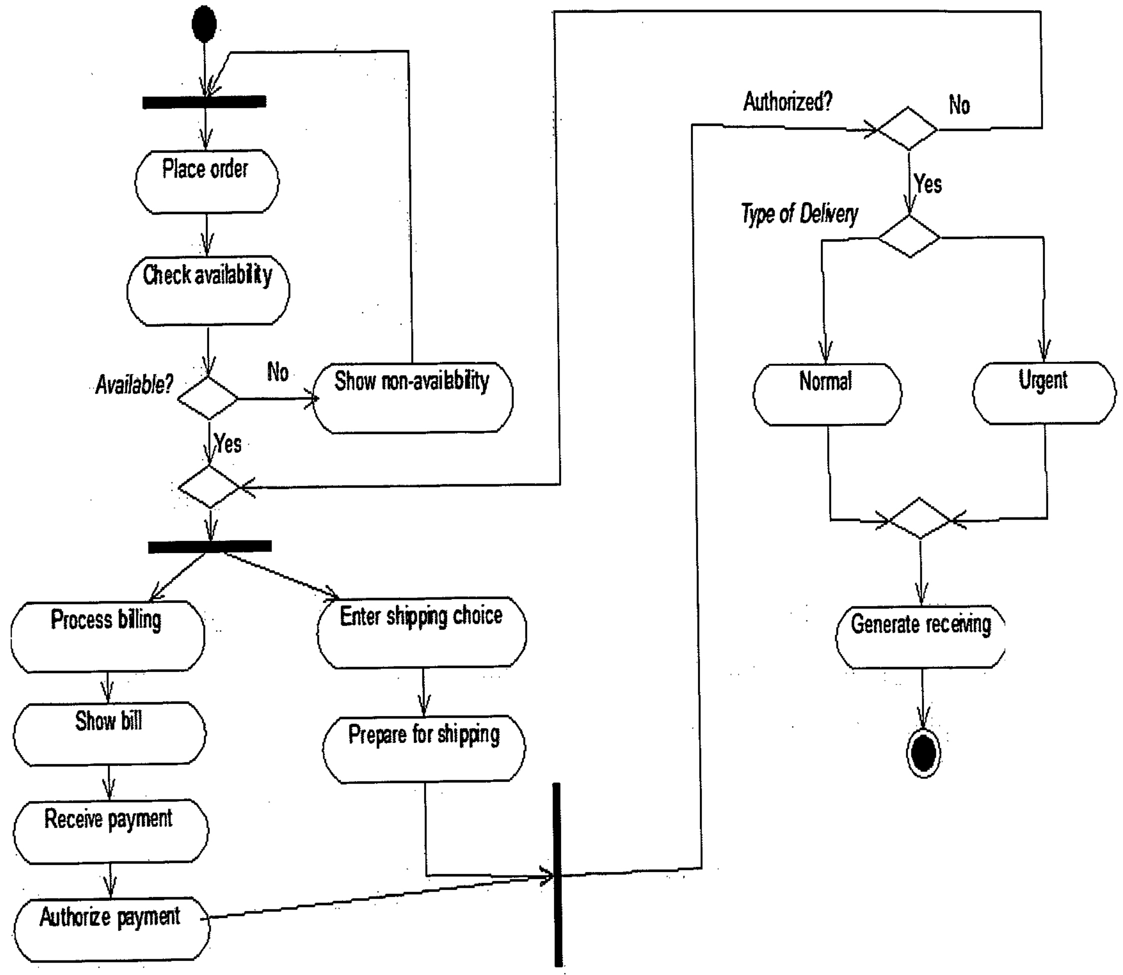


Figure 3 : Example Activity Diagram for Case Study

2.5.1 IOAD technique on generating test cases

By using IOAD technique, activity diagram is converted into a form where all the internal processing by the system is excluded [8]. The user interaction is emphasized in the new converted diagram as shown in Figure 4 where "I" stands for input and "O" stands for output.

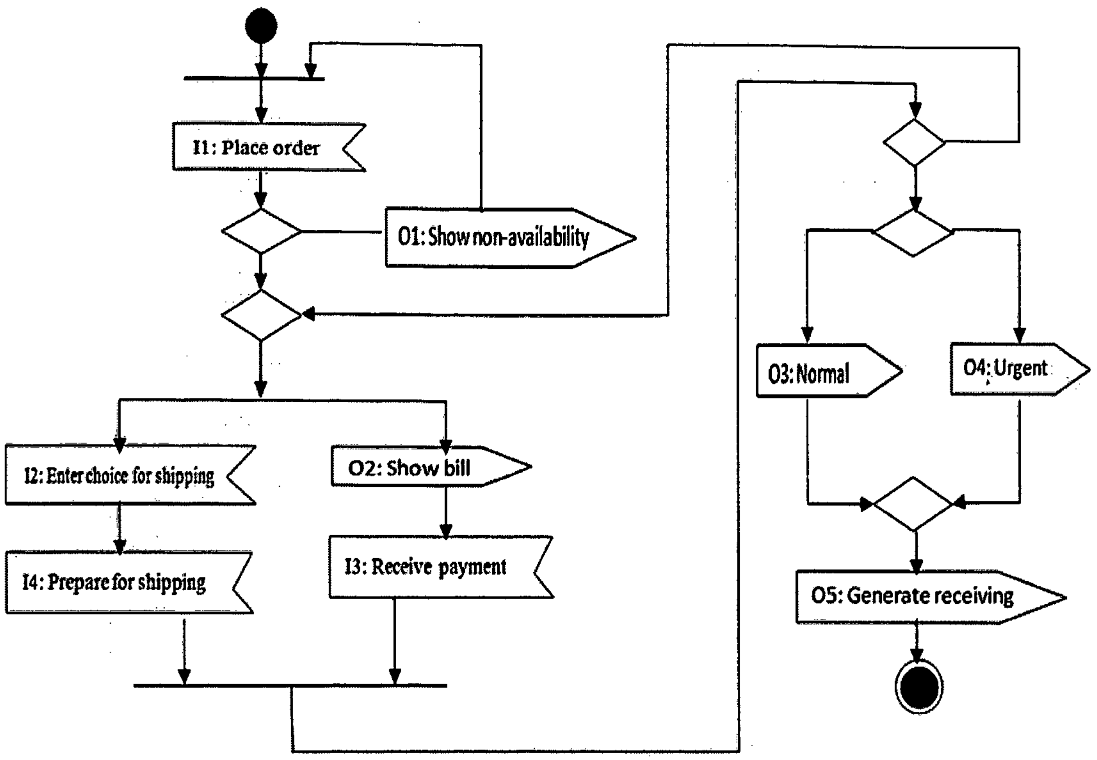


Figure 4 : Activity Diagram Based on IOAD

The following shows the four test cases resulted by using the single stimulus principle:

P1: I1-O2-I2-I3-I4-O3-O5

P2: I1-O1-O2-I2-I3-I4-O3-O5

P3: I1-O2-I3-I2-I4-O4-O5

P4: I1-O2-I3-I2-O2-I3-I2-I4-O4-O5

2.5.2 Condition-Classification Tree Method on generating test cases

Based on the steps in Figure 2, three decision points are extracted together with their guard conditions. The Condition-Classification tree table is drawn to mark the vertex that has relationship to each other as shown in Figure 5. The number of test case generated is shown at the left hand side of the table. In this example, four test cases are generated.

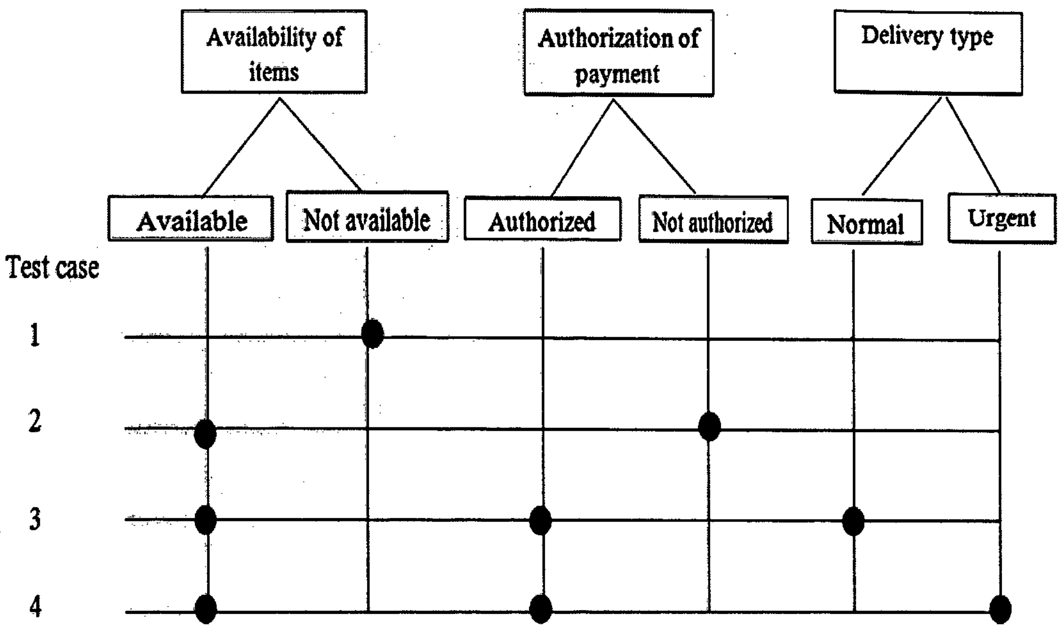


Figure 5 : Condition-Classification Tree Tables of Case Study

2.5.3 Sub activity diagram on generating test cases

From the case study, it is found that the activity "Check availability" can be further expands to more details. Therefore, the sub activity diagram is generated as shown in Figure 6.

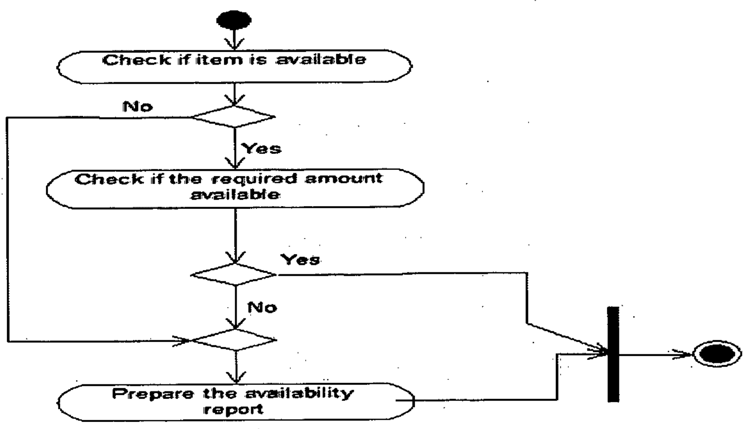

Check if item is available
No
Yes
Check if the required amount available
Yes
No
Prepare the availability report

Figure 6 : Sub Activity Diagram for "Check availability" Activity in Case Study

Three paths are generated from the sub activity diagram. By using all path combination technique for the four paths of ordinary activity diagram and three paths of

2.5.5 Enhanced technique on generating test cases

The enhanced technique first transfer all activities found in the activity diagram into an ADT as shown in Table 2. The ADT table records all the details of activity diagram including the merge, join, fork, and decision so as all functionalities and behaviour of the system are covered.

Table 2 : ADT of Case Study

Activity	No Reduction	Reduction
Join 1	A	
Place order	B	B
Check availability	C	C
Decision 1	D	
Show non-availability	E	E
Merge 1	F	F
Fork	G	
Enter shipping choice	H	H
Prepare for shipping	I	I
Process billing	J	J
Show bill	K	K
Receive payment	L	L
Authorize payment	M	M
Join 2	N	
Decision 2	O	
Decision 3	P	
Normal	Q	Q
Urgent	R	R
Merge 2	S	
Generate receiving	T	T

Return	U	U

The enhancement in this technique refers to the activities before and after reduction where the reduction done increases the readability of the ADG and at the same time all important functionalities from the ordinary activity diagram are remained. Another, the minimal test paths are obtained through reducing the generated test paths by combining each test path that will be used to test a loop. Each of the combined paths has the input, pre-conditions, post conditions, and the output [6]. Figure 8 and Figure 9 show the ADG before and after reduction respectively.

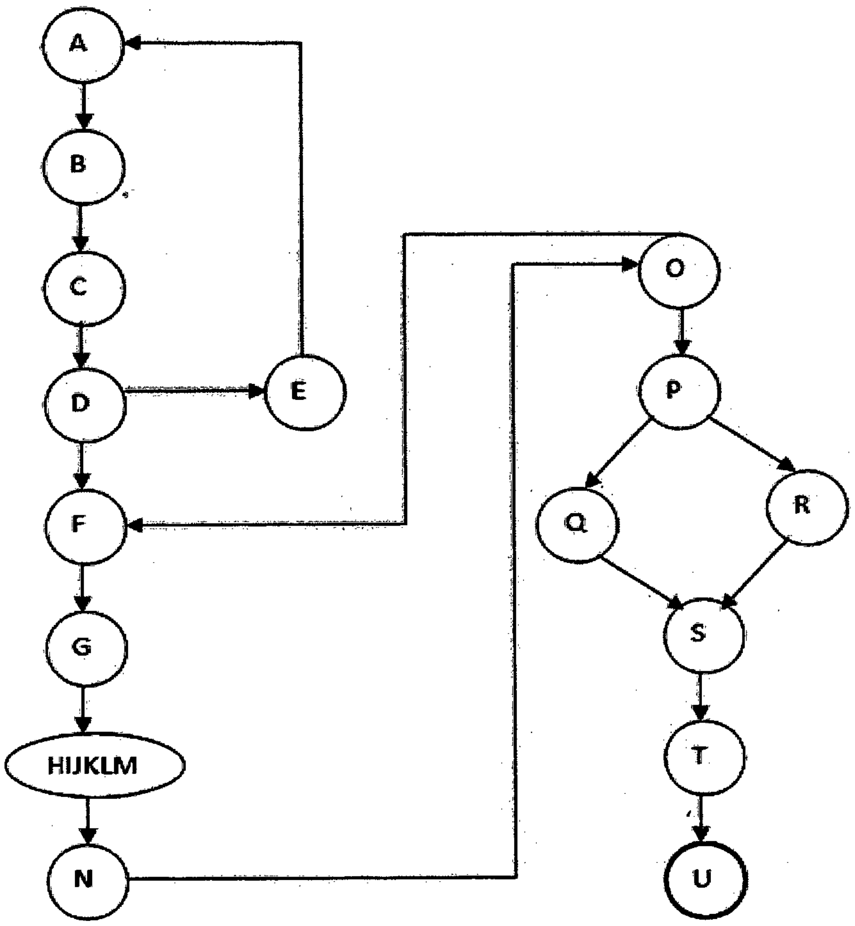


Figure 8 : ADG before Reduction of the ADT Table 2

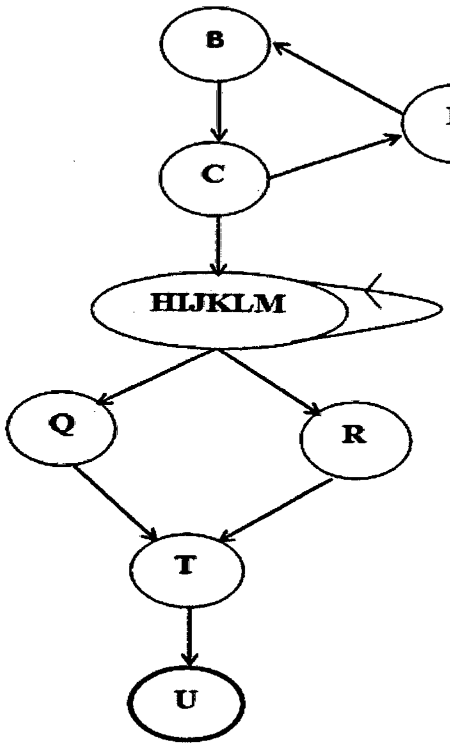


Figure 9 : ADG after Reduction of the ADT Table 2

By applying DFS on the Figure 9, four test paths are generated as following:

Test path 1: B-C-B-C-U

Test path 2: B-C-HIJKLM-HIJKLM-U

Test path 3: B-C-HIJKLM-Q-T-U

Test path 4: B-C-HIJKLM-R-T-U

From the generated test paths, minimization is done using reduction for loop technique. The final test paths number generated is two as following:

Test path 1: B-C-B-C-HIJKLM-Q-T-U

Test path 2: B-C- HIJKLM- HIJKLM-R-T-U