St. Cloud State University
theRepository at St. Cloud State

Culminating Projects in Computer Science and Information Technology

Department of Computer Science and Information Technology

6-2018

Selection of UML Models for Test Case Generation: A Discussion on Techniques to Generate Test Cases

Raj Kumar Rapolu
St. Cloud State University, rrapolu@stcloudstate.edu

Follow this and additional works at: https://repository.stcloudstate.edu/csit_etds

Part of the Computer Sciences Commons

Recommended Citation
Rapolu, Raj Kumar, "Selection of UML Models for Test Case Generation: A Discussion on Techniques to Generate Test Cases" (2018). *Culminating Projects in Computer Science and Information Technology*. 26.
https://repository.stcloudstate.edu/csit_etds/26

This Starred Paper is brought to you for free and open access by the Department of Computer Science and Information Technology at theRepository at St. Cloud State. It has been accepted for inclusion in Culminating Projects in Computer Science and Information Technology by an authorized administrator of theRepository at St. Cloud State. For more information, please contact rswexelbaum@stcloudstate.edu.

Selection of UML Models for Test Case Generation: A Discussion on Techniques to Generate Test Cases

by

Raj Kumar Rapolu

A Starred Paper

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree

Master of Science

in Computer Science

June, 2018

Starred Paper Committee:
Ramnath Sarnath, Chairperson
Jie H. Meichsner
Sneh Kalia

Abstract

Software development life cycle (SDLC) depicts the distinct phases a software needs to go through, starting off with the requirement engineering phase through the testing phase. Requirement engineering and testing hold a very important place in the development of the software. Testing plays such a crucial part in SDLC that more than development some projects spend more time on testing. This helps in finding bugs and ensure that a quality product is shipped.

Test case generation is the toughest step in the testing process. It involves a lot of effort to find errors in the code. To eliminate the tedious process of finding errors in the code and improve the efficiency of the software, an innovative approach called Model-Based Testing (MBT) has been evolved. In the MBT approach, testing begins at the design phase and is thus helpful in identifying the faults early. Unified Modeling Language (UML) has been used to generate test cases using MBT. There has been a lot of research and proposals made for MBT using different UML diagrams. But the problem with these approaches is that a single UML diagram like activity diagram or sequence diagram is not enough to generate test cases in all the scenarios. There might be situation where multiple models might be used to generate test case. This paper discusses about different existing MBT methodologies used for testing and discusses the guidelines for choosing an efficient UML diagram to generate test cases. It also discusses about the way of improving the models by using multiple UML models or by extending the models.

Keywords: Software testing, Model-Based Testing (MBT), Unified Modeling Language (UML).

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Software testing is one of the important steps of Software Development Life Cycle (SDLC). About 50% of the time taken to release a product is spent on testing [1]. Testing has helped organizations to release a reliable and stable product in to production. The important goal of testing is failure detection, which could be defined as finding out the differences between implementation and expected behavior based on specifications. As the systems are becoming large and complex, the need to find effective testing mechanism has become a vital part of the SDLC process. The testing process in SDLC depends mainly on four parts: test case generation, test case selection, test case execution, and test case evaluation [1]. All four parts are important but the part that takes a lot of effort and is vital to build an efficient product from day one is test case generation. It helps in identifying the test cases early in the software development so that errors could be identified early. Test case generation also needs a lot of effort and plays a key role on the efficiency and effectiveness of a software test [2].

As software becomes large and complex, the requirement to find a reliable test case generation scenario has taken a lot of importance. The existing methodologies of automated and manual testing are useful, but they fail to identify errors early in the SDLC. These methodologies help in identifying errors after the development process has started and does not identify errors at the design phase of the software. It is important to identify the test paths earlier in the software development life cycle as it helps the software developers in designing effective test cases that would cover all the possible scenarios [3]. This test path could be identified early by using a new software development paradigm, Model-Based Testing (MBT), by using the design models for testing software, especially object-oriented programs. Some of the important reasons for using design models in testing object-oriented programs are [4]:

i. Models help in testing the dynamic behavior of object-oriented systems, unlike the traditional testing techniques which consider only the static view of the code.
ii. Models help the software tester to understand the system in a better way and helps them to find the test information only by simple processing of models when compared to the use of code, which may become tedious and complex to test object-oriented programs.
iii. Model-based test generation helps in carrying out the development of the code and testing simultaneously by planning the generation of test cases early in the software development life cycle.

[5] defines MBT as "the automatable derivations of concrete test cases from abstract formal models, and their execution". MBT provides a more efficient way of testing as it provides a methodology with a combination of source code and system requirements for software testers to test the software. In MBT, test cases are generated using the models, which help the testers to understand the software in a better way and could get the test information with simple processing of the models [1]. One of the major advantage of MBT is its ability to detect errors from the early stage of development and generating test case without being dependent on any implementation of the design [6]. Using MBT, a model that describes about the system under test (SUT) could be used to generate test cases. Many of the MBT approaches use Unified Modeling Language (UML) models to visually depict the software systems.

Over the past few years, UML has been gaining a lot of popularity and has gained a lot of attention from researchers. UML offer different models such as use case, activity, and sequence diagrams to analyze the system. UML is flexible and is used to describe artifacts of a software system [7]. Each diagram in UML presents a view of the software and helps developers understand the system flow. Even though each type of UML diagram offers a view of the software there are some limitations to each type of model in generating test cases. The problem with the models is that each type offers unique features and is useful in certain scenarios but holds some limitation to generate test cases when they are used in a different scenario. To explain this in detail, let us consider an example of sequence diagram which is good in capturing the messages between objects, but it is alone not enough to decide different components such as input, expected output, pre and post conditions of the test cases. If we consider use case diagram, as it is used to review the requirements it cannot be used for test case generation [1]. Some of the UML diagrams does not support looping and iteration and cannot be used in such features.

As each of the UML diagram has limitations, it is difficult to identify and select a UML diagram that can be used for distinctive features of case study. In this paper, we would discuss this problem, different UML diagrams and how they could be used in test case generation. We would discuss how a single or a combination of UML models can be used to generate test cases. This discussion would help researcher and testers find out the models that can be used for a scenario of test cases generation.

1.1 Objectives

The main objective of this paper is to discuss the existing techniques which use the UML diagram to generate test cases. Different UML diagrams can be used to generate test cases, but it is difficult to say which would be a better option for case study with a set of features. For Example: Sequence diagram may not be a good option to generate test cases when the case study has a looping activity. In such cases, choosing two or more UML diagrams or an extension of UML diagram would help in covering all the possible test case paths.

In this paper, we would discuss the techniques which use Activity diagram, Sequence diagram, and State chart diagram to generate test cases. We would consider three case studies, apply these techniques and discuss about the advantages and limitations of choosing a UML diagram for a case study.

Finally, we would discuss the technique proposed in [1] to overcome the limitations of sequence diagram when the case study has a looping activity and also discuss the guidelines proposed in [1] which helps in selecting one or more UML diagram to generate test cases.

1.2 Paper Contents

In this paper, we discuss techniques that make use of UML diagrams to generate test cases. After the discussion on different techniques, we compare the test case generated by them, discuss the drawbacks of the UML diagrams for generating test cases and discuss the guideline to overcome the drawbacks while using state chart diagram to generate test cases.

Chapter 1 presents introduction which discusses about the importance of model-based testing and its advantages. Section 1.1describes the objectives of the paper. Chapter 2 discusses the related work in the field of model-based testing. Chapter 3 presents the case studies that are used to implement the methods. We use two case studies Grocery Item Buying System (GIBS) and Grocery Item Request (GIR) to discuss the techniques. Section 3.2 discusses the techniques in detail. The techniques discussed in this paper use activity, sequence or state chart diagram.

Chapter 3 also discusses the results obtained by comparing the case studies. Chapter 4 describes about the guideline that can be used to eliminate the drawbacks while using state chart diagram to generate test cases. Chapter 4 makes use of Grocery Item Buying System (GIBS) case study to describe the implementation of the guideline. Chapter 5 describes the conclusion of the paper. It presents the overview of the results discussed. And the final section contains the references to articles, books which have been used during the study of this paper.

Chapter 2: Related Work and Notation

In this chapter we will discuss about some of the related work in this field and the techniques that research have used to generate test cases using MBT. We will also discuss notation that we will be using through the paper.

2.1 Related Work

The approach proposed in [8] generates the test cases by applying a genetic algorithm on dominance tree, which is generated from the control flow graph. In a paper by [9], the authors presents the approach of generating test cases by making use of UML class and sequence diagrams. In the paper by [10], the authors propose a prototype tool which helps in deriving the test cases using activity diagrams. The paper presented by [11] proposes the use of System Modeling Language (SysML) activity diagrams in combination with Modeling and Analysis of Real-Time and Embedded systems (MARTE) to validate the requirements at early phase of software development life cycle. It also presents an approach for the translation of SysML activity diagram to Time Petri Net with Energy constraints (ETPN). The technique proposed in [4] presents a graph-based method to generate test cases from individual activity diagrams.

In another study which is proposed in [12] makes use of sequence diagram to generate the test cases. They convert the sequence diagram to sequence diagram graph (SDG) and generate the test cases from SDG. Each node in SDG has information collected from use cases, class diagram and data dictionary. Then, test cases are generated by covering all the possible paths from initial node to final node which would cover all the interactions and message sequence paths. Automatic Teller Machine (ATM) case study with ATM pin validation as the use case was considered to implement the approach.

Besides these research studies, there have been other studies which used a combination of UML diagrams to generate test cases. [13] proposed a technique which uses sequence diagram and state chart diagram to generate test cases. The selection of these UML diagrams is based on the reasons that sequence diagram describes the interaction between the components whereas state chart diagram shows the transitions between states in UML. In this technique the sequence diagram is converted in to sequence graph and the state chart diagram is converted in to state chart graph, and ultimately these two techniques are combined to create system testing graph (SYTG). And then, a genetic algorithm is applied to this graph to optimize test cases automatically based on coverage criteria and fault model. They have used Online voting system as the case study and have proved that by combining multiple UML diagrams maximum number of test cases could be covered and helps in solving faults like scenario faults, error handling, and other faults in the system.

The research study proposed in [14] made use of a combination of activity diagram and sequence diagram. The activity diagram is converted in to activity graph and sequence diagram is converted in sequence graph. Then, these two graphs are integrated to form a system graph which could be used to generate test cases. The depth-first search (DFS) is used to traverse deeper in the graph. This approach is capable of handling state explosion problem in case of concurrent system [1]. An example of ATM card validation is used to implement this technique.

The work carried out by [15] uses UML state diagram and Object Constrained Language (OCL) to generate test cases. They convert the state diagram into Finite State Machine (FSM) where each node represents state and the arrows describe the transition between the states. FSM stores valuable information collected from use case diagram including pre and post conditions that are expressed using OCL. To generate the test cases, FSM is traversed based on three types of coverage criteria which are transition coverage, transition pair coverage and provides state coverage. For their case study, they have taken the example of ATM cash transaction problem from ATM case study.

From the literature review, we could observe that there are different methods available to generate test cases using different UML diagrams. The case studies selected by the researchers are also mainly concentrated on Online Voting System, Conference Management System, and Automatic Teller Machine System. Every research has used its own method of test case generation using different UML diagrams. Some research studies use a single UML diagram where as other use a combination of UML diagrams. Most of the researches have used a technique of converting the UML diagram into an intermediate graph which is used to generate test cases. The graph is traversed based on coverage criteria and fault model to generate test cases. In some studies, the researchers use a combination of UML diagrams as sequence diagram and state chart diagram are not alone sufficient to generate test cases. Because of this limitation an integration of two UML diagrams is required in some cases.

2.2 Notation

There are many notations that have been used throughout the paper. This section will help us understand all the notations in detail by using simple examples.

Considering the simple use case of Automatic Teller Machine Withdrawal (ATMW), we explain the different notations that have been used throughout the paper.

Use Case of ATMW: The user selects the withdrawal option from the list of options presented on the screen. The system requests the user to insert the card. Once the user inserts and removes the card, it validates the card to check if it supported by the system. If it does not support, it displays the error text and exit. If the system supports the withdrawal from the card, the system requests the user to enter a pin. The system allows user to enter incorrect pin three times. If the user has entered a valid pin, then the system will present the amount that is available in the account and asks to enter the amount to be withdrawn. If the amount entered is below the available amount, it returns the receipt and money. But if it is greater than the available amount, it displays an error and exits the system.

Unified Modeling Language popularly known as UML is a language for describing the software system visually using models. A software might contain a lot of modules with a huge code base. To provide an abstract view of the system architecture, transfer the knowledge quite easily and to understand the software system quickly and efficiently, UML provides models which describe the structure of a software system. Activity, state chart, sequence and use case diagram are some of the most important models of UML.

Activity Diagram represent the systematic flow of the system. It visually presents the series of actions performed on the system by the user. It is similar to flow chart. It has a start and end. Arrows describe the flow of system, whereas rectangle boxes represent the actions performed. Diamond box represents the condition execution. Figure 1 presents the activity diagram for ATMW case study.

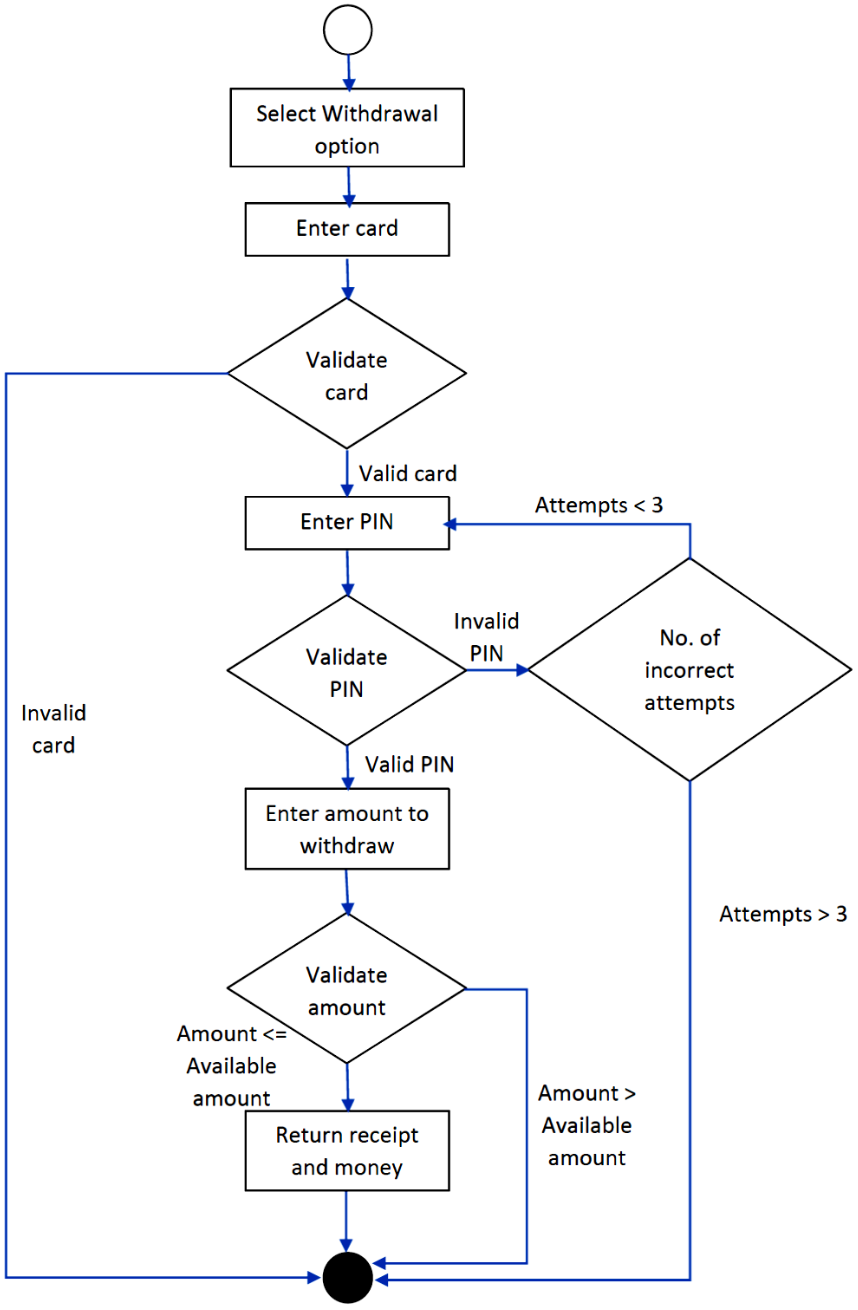

Figure 1: Activity Diagram for ATMW

Sequence Diagram represents the interaction between objects in the system. It describes the messages between the object in a time sequence and presents the name of the classes involved in the interaction. Figure 2 presents the sequence diagram for ATMW case study. The rectangular boxes represent the classes in the system. It displays the interaction between the components and actor being one of the components in the system. For Example: message m1 presents the interaction between the actor and ATMW View and the dotted arrows represent the response from the system. The blocks are the conditional blocks that would get executed when a condition is true. For Example: “Amount > Available Amount” is one of the condition presented in the box. The condition c5 and the message m16 only get executed when the condition is satisfied.

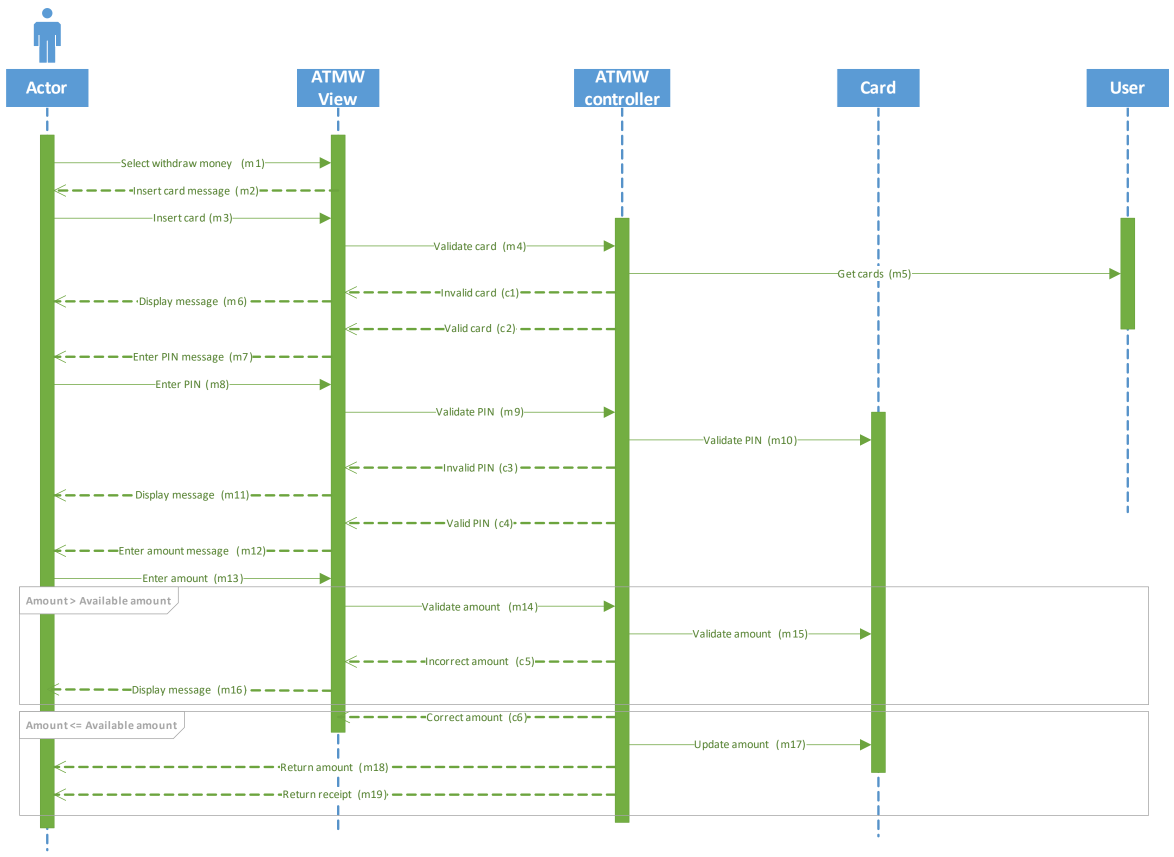

Figure 2: Sequence Diagram for ATMW

State Chart Diagram represents the dynamic nature of a system. It presents the different states the system enters during its lifetime. The transition between states are triggered by the events. Figure 3 presents the state chart diagram for ATMW case study. The black circle represents the start of the use case and rectangular box represents the states of the use case. Display options, insert card, validate card etc., are different states of a software system. The arrow represents the action on the state and its head points to the state the software attains due to the action. For Example: “Select Withdraw” is the action performed on the “Display options” state which moves the system to the new state “Insert card”.

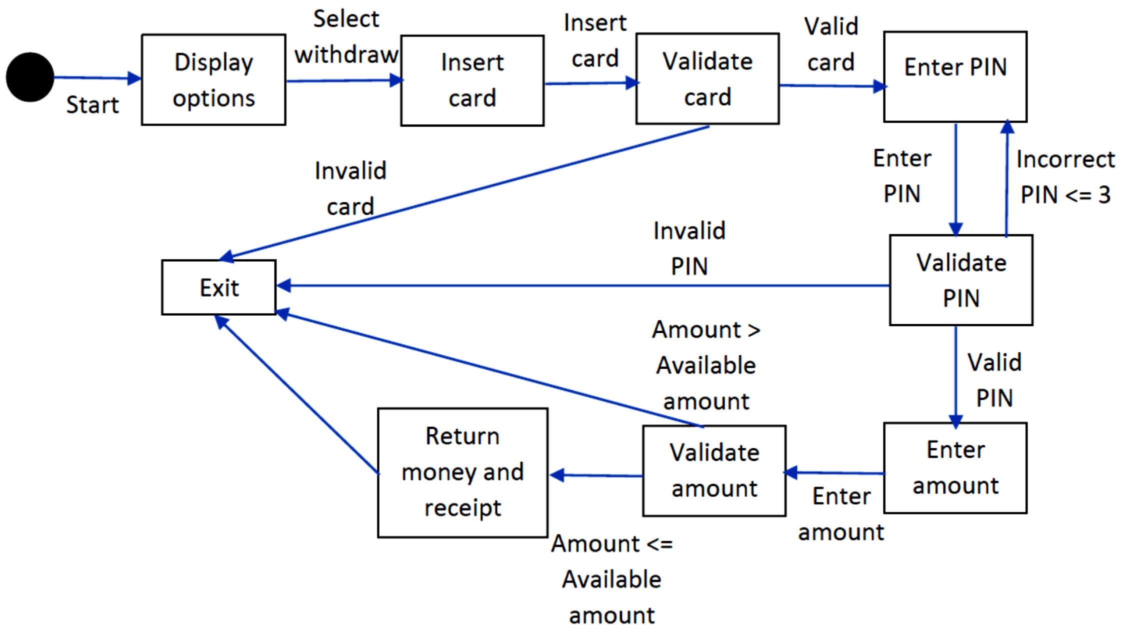

Figure 3: State Chart Diagram for ATMW

Activity Dependency Table (ADT) presents the dependencies of nodes, their input and output. It helps in gathering information regarding the nodes of activity diagram. Table 1 presents the ADT of GIBS case study. It has five columns symbol, activity name, dependency, input, and output.

Symbol represents the activity in an activity diagram

Activity Name is name of the activity in an activity diagram

Dependency represent the dependency that the activity has on other activities

Input describes the input for the activity

Output describes the output from the activity

For Example: For the second row in ADT, “B” represents the symbol for activity name “Enter Card”. It has dependencies on node A with an input of “Withdraw option selected: and an output of “Card Entered”.

Table 1: Activity Dependency Table (ADT) for ATMW

Symbol	Activity Name	Dependency	Input	Expected Output
A	Select withdrawal option		Select withdraw	Selecting withdraw option
B	Enter Card	A	Withdraw option selected	Card Entered
C	Validate Card	B	Card Entered	True (Card = “Valid”) False (Card = “Invalid”)
D	Enter PIN	C, E	True (Card = “Valid”)	PIN Entered
E	Validate PIN	D	PIN Entered	True (PIN = “Valid”) False (PIN = “Invalid”)
F	No. of Correct Attempts	E	False (PIN = “Invalid”)	Enter PIN (Attempts < 3) Exit (Attempts >= 3)
G	Enter Amount to Withdraw	E	True (PIN = “Valid”)	Entered amount
H	Validate Amount	G	Entered amount	True (Amount = “Valid”) False (Amount = “Invalid”)
I	Return receipt and money	H	True (Amount = “Valid”)	Display receipt and money
J	Exit	C, F, H, I	Display receipt and money	Exit the system

Activity Dependency Graph (ADG) is generated from ADT. It presents the dependency of node on another visually and helps in generating the test case paths. Figure 4 presents the ADG generated for GIBS case study. The arrows in ADG depict the dependency of a node on another. The information required for ADG are taken from ADT. ADG can also be defined as visual representation of ADT.

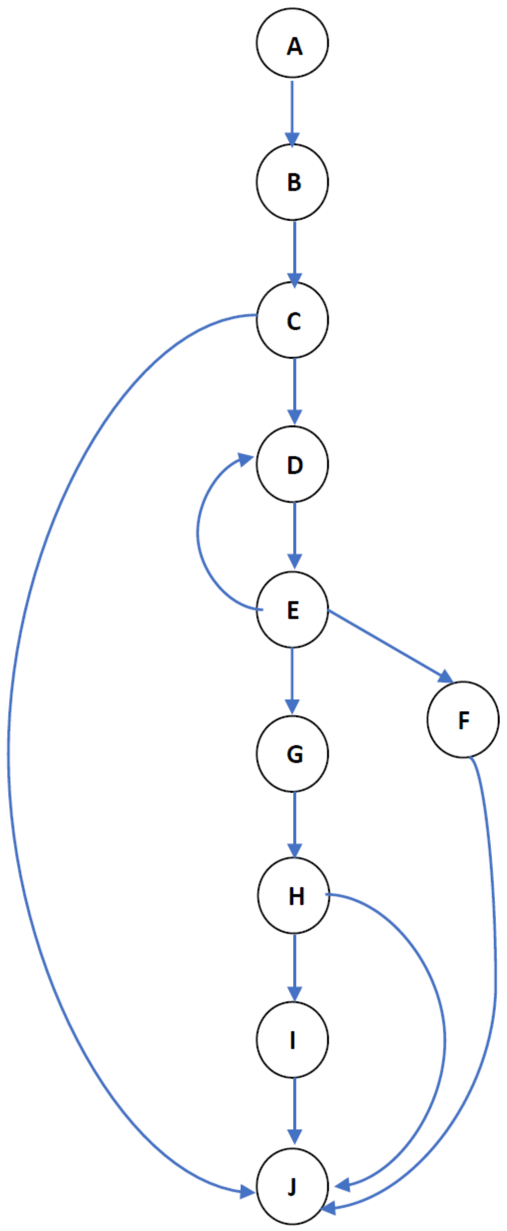

Figure 4: Activity Dependency Graph (ADG) for ATMW

Sequence Diagram Graph (SDG) consists of nodes and edges where each node defines the event and edge represents the transition between the events. It covers the transitions between the events and contains all the information to generate test cases.

State Transition Graph (STG) describes the process of the system using the vertices. STG is generated from state chart diagram by representing the state of a system using a vertex (V). STG contains a list of vertices which can be defined as Vi = (V1, V2, V3….) and there is a unique edge which joins two vertices. The edges represent transition between the states or vertices. Figure 6 presents the STG for GIBS case study.

Coverage Criteria can be defined as guidelines or requirements that need to be met to generate test cases. They help in covering all possible test path. Node, all-transition and all-transition coverage criteria are some of the coverage criteria helpful in deriving test cases.

Chapter 3: Case Studies and Discussion on Techniques

In this section, we would discuss case studies that we would be using in the paper and then discuss different techniques of MBT using activity, sequence, and state chart diagram.

3.1 Case Studies

The two case studies that we would be using to explain techniques which use activity, sequence and state chart diagram are Grocery Items Buying System (GIBS) and Grocery Item Request (GIR). There were different case studies used by many researchers to explain MBT. Some of case studies researchers have used are Automatic Ticket Machine System (ATMS), Automatic Teller Machine Pin Authentication (ATMPA), Automatic Teller Machine Withdrawal (ATMW) etc. GIBS and GIR are functionally so different and involve a different industry of use cases. ATMS and ATMPA case studies were used with a looping activity whereas the ATMW case study was used without any looping activity. When the case studies are compared based on the looping activity, GIBS case study involves looping activity whereas GIR case study does not involve any looping activity. Thus, both GIBS and GIR helps us in discussing the techniques and the results they produce when case studies with varying features are used to generate test cases. This is the primary reason to choose these case studies, to study how different techniques work for a case study involving looping activity and the one that does not involve looping activity.

3.1.1 Grocery Items Buying System (GIBS)

The precondition to this user story is that the user has successfully logged in to the system. The user selects a store and clicks on continue. If the user has not selected any store, the user will see a message to select the store and displays the list of stores. If the store is valid he will be presented with a list of items available in the store and he can select them. The selected items are stored in the cart. Then the user checkouts the cart and system searches for the number of items in the cart. If the selected items are zero, then the user is presented with an option to select items or exit the system. If user selects to exit he will return to the initial screen otherwise the system presents the list of items to the user. If it is greater than zero, system will check if all the items and their quantity are still available. If the items are available, the system request for payment details. If not available, it corrects the items to available items and then requests to enter payment details. Once the user enters payment details, it authorizes the payment. After the authorization the user will get a confirmation coupon and then he can go and pick it up at the store. Otherwise, he will be prompted with a message saying that the payment method has failed, and the user needs to correct the payment details.

3.1.2 Grocery Item Request (GIR)

The user wants to request a grocery store about the item he is interested in buying with them which the store has not stocked. The user enters the item name and system searches for all the items matching the name. If there is no match the system exits with a message. If the item exists, the list of items which matches the name are displayed. The user selects an item and the system searches if a request already exists for that item. If it exists, the existing request is returned otherwise it creates a new request and returns the request.

3.2 Techniques

We use GIBS and GIR case study to implement the techniques and generate test cases. The techniques we are going to discuss make use of one among the activity, sequence or state chart diagram. We are going to discuss about why a technique generate specific set of test cases and the disadvantages of using a model to derive test cases.

3.2.1 Test Case Generation Using Activity Diagram

[16] used activity diagram to generate test cases. They have used ATM withdrawal case study to propose test case generation technique. In this test case generation technique proposed by [16], they have used activity diagram as a source of test case generation. The reason they have selected the activity diagram is that the activity diagram helps in presenting the activity flow of the case study. In this section, we discuss about [16] technique with the help of a case study Grocery Items Buying System (GIBS).

Figure 5 presents the activity diagram created for GIBS. It presents how the users interact with the system and presents the flow of various activities between user and GIBS. In the activity diagram, each node represents the activity or action that needs to be carried out where as the edges represent the transition between the activities.

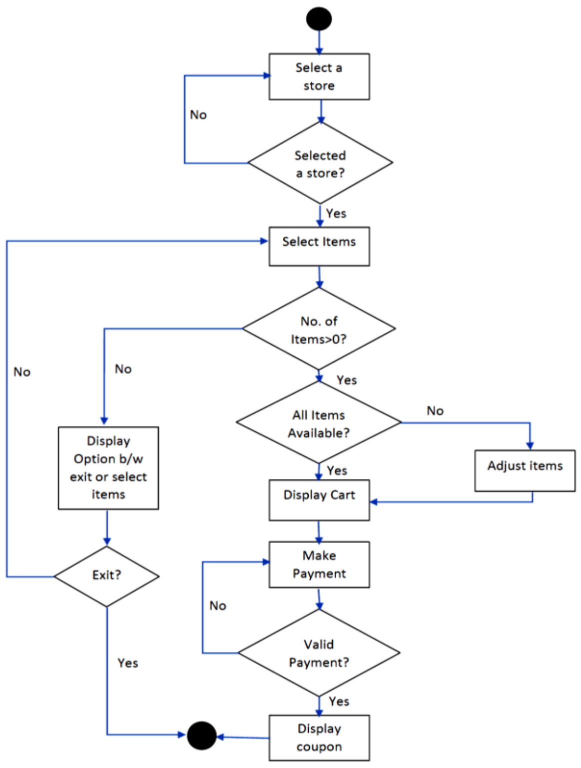

Figure 5: Activity Diagram for GIBS

3.2.1.1 Technique Overview

The technique proposed by [16] uses activity diagram for test case generation. Using activity diagram, an Activity Dependency Table (ADT) is created. In ADT, each activity is given a symbol, and contains the information about the expected input and output, dependencies on other activities. After the creation of ADT, it is converted in to Activity Dependency Graph (ADG) from which the test cases are generated.

3.2.1.2 Generation of Activity Dependency Table (ADT)

After the creation of activity diagram, an ADT is created which contains information about the activities and its unique symbol, input and output, and dependencies on the other activities. Table 2 presents the ADT for GIBS case study.

Table 2: Activity Dependency Table (ADT) for GIBS

Symbol	Activity Name	Dependency	Input	Expected Output
A	Select a Store		Select store	Store Selected
B	Validate Store	A	Store Selected	True (Store = “Selected”), False (Store = “Not Selected”)
C	Select Items	B	True (Store = “Selected”)	Items Selected
D	Calculate No. of Items	C	Items Selected	True (Items>0), False (Items=0)
E	Display Option	D	Items = 0	Display Options
F	Choose Exit or select items	E	Options Displayed	True (Exit), False (Select items)
G	Verify if all items are available	D	Items>0	True (All items available) False (All items not available)
H	Adjust items	G	All items not available	Adjusted items
I	Display Cart	G, H	Select available items	Displays items and price
J	Make Payment	I, K	Display items and price	Processes payment request
K	Validate Payment	J	Processes payment	True (Valid payment), False (Invalid payment)
L	Display Coupon	K	Valid payment	Displays coupon
M	Exit	L, F	Display coupon or exit	Systems becomes Idle

3.2.1.3 Generation of Activity Dependency Graph (ADG)

In the next step, ADT and activity diagram are used to generate ADG. ADG is made up of nodes and edges. Each node is represented by the symbol used to denote each activity and edges represent the transition between activities or nodes. An edge with a pointing arrow is drawn from one node to the other node that it is dependent on. The arrow points out the direction of the dependency. By applying this procedure to the entire ADT, we create an ADG. Figure 6 illustrates ADG for GIBS.

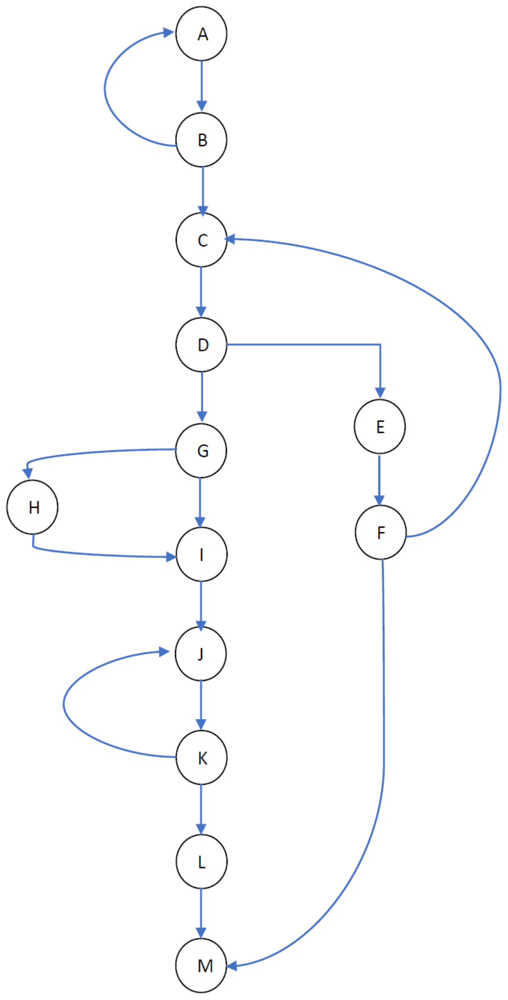

Figure 6: Activity Dependency Graph (ADG) for GIBS

3.2.1.4 Generation of Test Cases

After the creation of ADG, all the possible test case paths are generated using ADG. Using the Depth First Search (DFS) technique, the test case paths are generated. By analyzing and using DFS for ADG presented in Figure 2, we could obtain six possible test case paths. The possible test case paths are presented below:

Path 1: A-B-A-B-C-D-E-F-M

Path 2: A-B-A-B-C-D-G-I-J-K-L-M

Path 3: A-B-C-D-E-F-C-D-G-I-J-K-L-M

Path 4: A-B-A-B-C-D-G-H-I-J-K-L-M

Path 5: A-B-C-D-E-F-C-D-G-H-I-J-K-L-M

Path 6: A-B-C-D-G-I-J-K-J-K-L-M

After generating the test paths by traversing through ADG, the information from ADT is extracted and added to test path. A table is created by adding the information and is presented in Table 3. The table lists all the six paths that are generated along with all the information regarding the activities. The number of test cases generated using this technique are same when compared with the state chart diagram technique proposed in [17].

Table 3: Test Cases for GIBS

Test Case	Test Path	Node Input	Node Expected Output	Test Case Input	Test Case Expected Output
1	A B A B C D E F M	Select Store False (Store = “Not Selected”) Select Store True (Store = “Selected”) Select Items Calculate No. of Items (Items = 0) Display Option Choose Exit or select items Exit	False (Store = “Not Selected”) True (Store = “Selected”) Items = 0 Exit	Items = 0	Failure, Exit system

2	A B A B C D G I J K L M	Select Store False (Store = “Not Selected”) Select Store True (Store = “Selected”) Select Items Calculate No. of Items (Items>0) Verify if all items are available (All available) Display Cart Make Payment Validate Payment True (Valid payment) Display Coupon Exit	False (Store = “Not Selected”) True (Store = “Selected”) Items>0 All Items available True (Valid payment) Get coupon	Valid payment	Get Coupon
3	A B C D E F C D G I J K L M	Select Store True (Store = “Selected”) Select Items Calculate No. of Items Items = 0 Display Option Choose Exit or select items Select Items Calculate No. of Items Items > 0 Verify if all items are available All Items are available Display Cart Make Payment Validate Payment True (Valid payment) Display Coupon Exit	True (Store = “Selected”) Items = 0 Item > 0 All Items are available True (Valid payment) Get Coupon	Valid payment	Get Coupon
4	A B A B C D G H I J K L M	Select Store False (Store = “Not Selected”) Select Store True (Store = “Selected”) Select Items Calculate No. of Items Items > 0 Verify if all items are available All items not available Adjust items Display Cart Make Payment Validate Payment Valid payment Display Coupon Exit	False (Store = “Not Selected”) True (Store = “Selected”) Items > 0 All items not available True (Valid payment) Get Coupon	Valid Payment	Get Coupon

5	A B C D E F C D G H I J K L M	Select Store True (Store = “Selected”) Select Items Calculate No. of Items Items = 0 Display Option Choose Exit or select items Select Items Calculate No. of Items Items > 0 Verify if all items are available Adjust items Display cart Make Payment Validate Payment Valid payment Get Coupon	True (Store = “Selected”) Items = 0 Items > 0 Validate Payment Get Coupon	Valid Payment	Get Coupon
6	A B C D G I J K J K L M	Select Store Validate Store True (Store = “Selected”) Select Items Calculate No. Items Items > 0 Verify if all items are available All items are available Display Cart Make Payment Validate payment Invalid payment Make payment Validate payment Valid payment Get coupon	True (Store = “Selected”) Items > 0 All items are available Invalid payment Valid payment Get coupon	Valid Payment	Get Coupon

3.2.2 Test Case Generation Using Sequence Diagram

In this section we use the technique proposed by [12] which uses sequence diagram to generate test cases. The technique proposed by [12] uses a case study, Automatic Teller Machine Pin Authentication (ATMP). Sequence diagram helps in picturizing the interaction between the user and the system. In this section, we use Grocery Items Request (GIR) case study to present this technique and compares the test case generated in this case study with other case studies.

Figure 7 presents the sequence diagram created for GIR. It presents the actions and messages exchanged between the user and the system.

3.2.2.1 Technique Overview

In this technique, first, a sequence diagram is created which would capture the interaction between the user and the system. Then an operation scenario is used to convert sequence diagram in to Sequence Diagram Graph (SDG). Using SDG, test cases are generated.

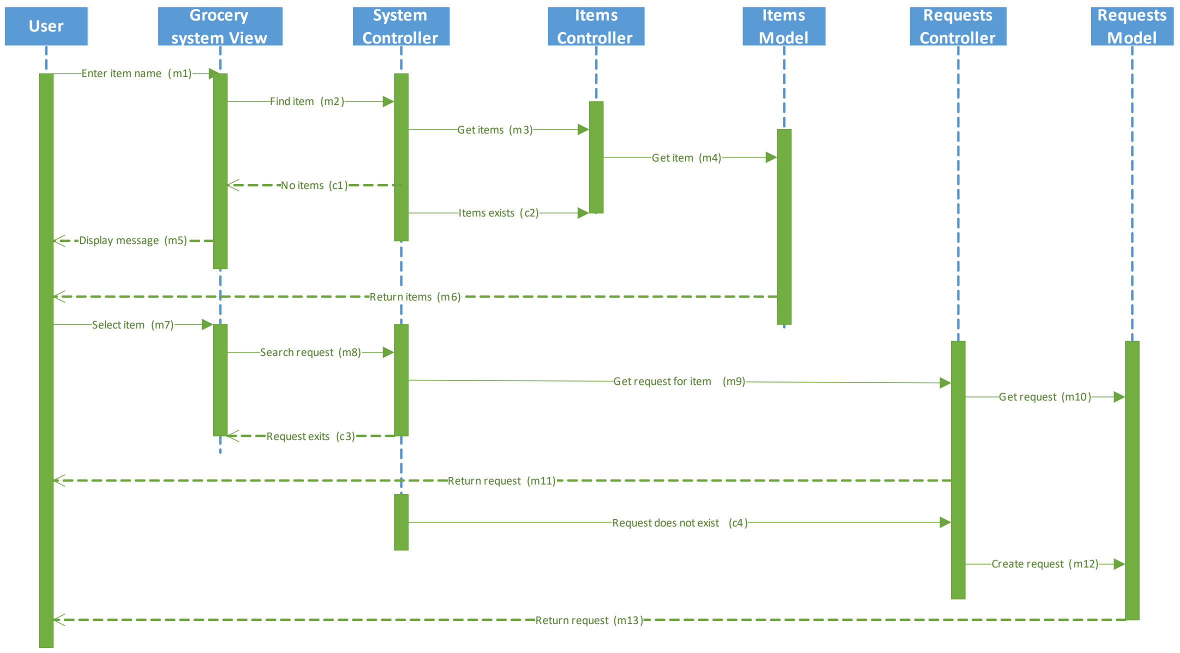

Figure 7: Sequence Diagram for GIR

3.2.2.2 Generation of Operation Scenario

[12] proposed operation scenario as the guide to convert sequence diagram to SDG. They have defined operation scenario as a quadruple and is stated as below:

aOpnScn: <ScnId; StartState; MessageSet; NextState>

Where,

ScnId: A unique number which identifies each operation scenario.

StartState: It is the starting point of a scenario.

Message set: Represents all the events that occur in an operation scenario.

NextState: It is the next state the system enters after the completion of the scenario.

All the events that occur in a scenario are represented as a tuple:

<messageName; fromObject; toObject[|guard]>

Where,

messageName is the name of the message with its signature

fromObject is the sender of the message

toObject is the receiver of the message

|*guard* parameter is the condition to subject with the event occurs.

An operation scenario can have only a single startState but can have multiple endState. The operation scenarios for GIR case study are presented in the Table 4. State X is defined as the start state for all the scenarios, whereas State Y as the failure state and State Z as the success state for all the operations.

Table 4: Operation Scenarios for GIR

<Scn 1	<Scn 2	<Scn 3
State X	State X	State X
S1: (m1, a, b)	S1: (m1, a, b)	S1: (m1, a, b)
S2: (m2, b, c) | c1	S2: (m2, b, c) | c2	S2: (m2, b, c) | c2
S3: (m3, c, d)	S3: (m3, c, d)	S3: (m3, c, d)
S4: (m4, d, e)	S4: (m4, d, e)	S4: (m4, d, e)
S5: (m5, b, a)	S6: (m6, d, a)	S6: (m6, d, a)
State Y	S7: (m7, a, b)	S7: (m7, a, b)
	S8: (m8, b, c) | c3	S8: (m8, b, c) | c4
	S9: (m9, c, f)	S9: (m9, c, f)
	S10: (m10, f, g)	S10: (m10, f, g)
	S11: (m11, f, a)	S12: (m11, f, g)
	State Z	S13: (m13, f, a)
		State Z

3.2.2.3 Transformation of Sequence Diagram in to Sequence Diagram Graph (SDG)

After all the operation scenarios are identified, an SDG is created. An SDG consists of nodes and edges where each node defines the event and edge represents the transition between the events. The SDG created using the sequence diagram and operation scenario is presented in Figure 8.

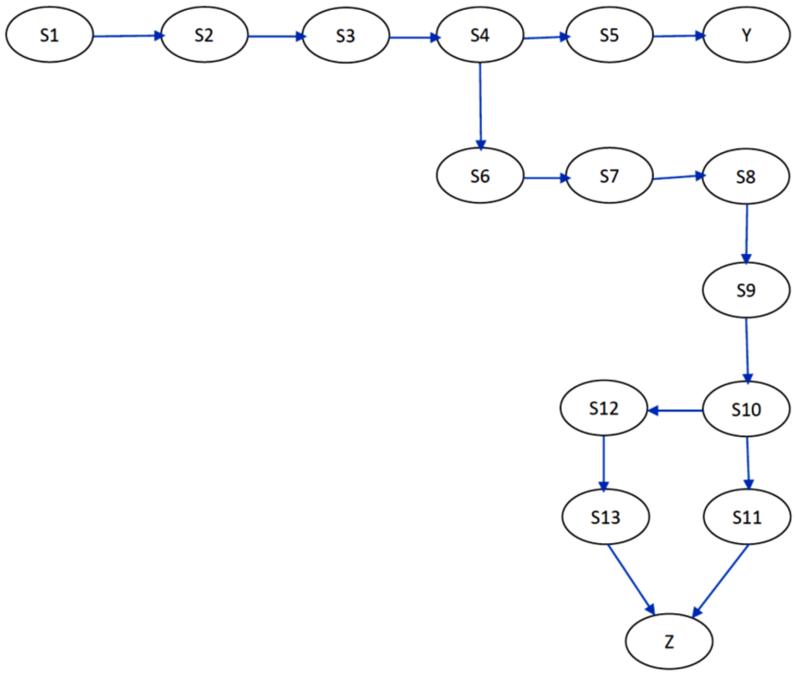

Figure 8: Sequence Diagram Graph (SDG) for GIR

SDG contains all the information required to generate test paths. It covers all the interactions starting from start node to the final node. There are two different final nodes, final node Y and final node Z. The final node Y denotes an unsuccessful transaction whereas final node Z denotes the successful transaction. The test case that are generated from SDG are:

Path 1: S1-S2-S3-S4-S5 = Unsuccessful

Path 2: S1-S2-S3-S4-S6-S7-S8-S9-S10-S11 = Successful

Path 3: S1-S2-S3-S4-S6-S7-S8-S9-S10-S12-S13 = Successful

These test path can be used to generate the final test cases. Each test path contains all the information including the expected input and output.

3.2.2.4 Generation of Test Cases

A set of test sets are defined to generate test cases. This sets could be used to detect any faults that occur when an object invokes a method of another object or whether message passing follows the correct sequence to finish an operation. The coverage criteria for this technique is defined as, given a test set A and sequence diagram B, T must cause each sequence of message path exercised at least once [1]. This coverage criteria and fault model is used to generate the test cases. The generated test cases are presented in Table 5.

Table 5: Test Cases for GIR

Test case scenario	**Input**	**Output**
1	Product Name = “Invalid”	Message would be that there are no items with the entered name
2	Product Name = “Valid” Request = “Exists”	Displays the existing request
3	Product Name = “Valid” Request = “Not exists”	Creates and displays request

The three test cases generated using this technique are same when compared to the test cases generated using [16] which uses activity diagram to generate the test cases. The technique proposed in [17] uses state chart diagram to generate test case. When [17] technique is used, the test cases generated are consistent with that of the other two techniques. So, for GIR case study the test cases generated are all the same in all the techniques and this is because the GIR case study does not contain any looping activity. This confirms that the features of the case study affect the generation of test cases.

3.2.2.5 Test Case Generation for GIBS Using Sequence Diagram

[12] used sequence diagram to generate test cases. Here, we use GIBS case study which involves looping activity to generate test cases.

Using sequence diagram, we generate the operation scenario table and then we create SDG from the operation scenario table.

Figure 9 presents the sequence diagram for GIBS case study. The operation scenario for this case study is presented in Table 6. It produces two unsuccessful and two successful scenarios.

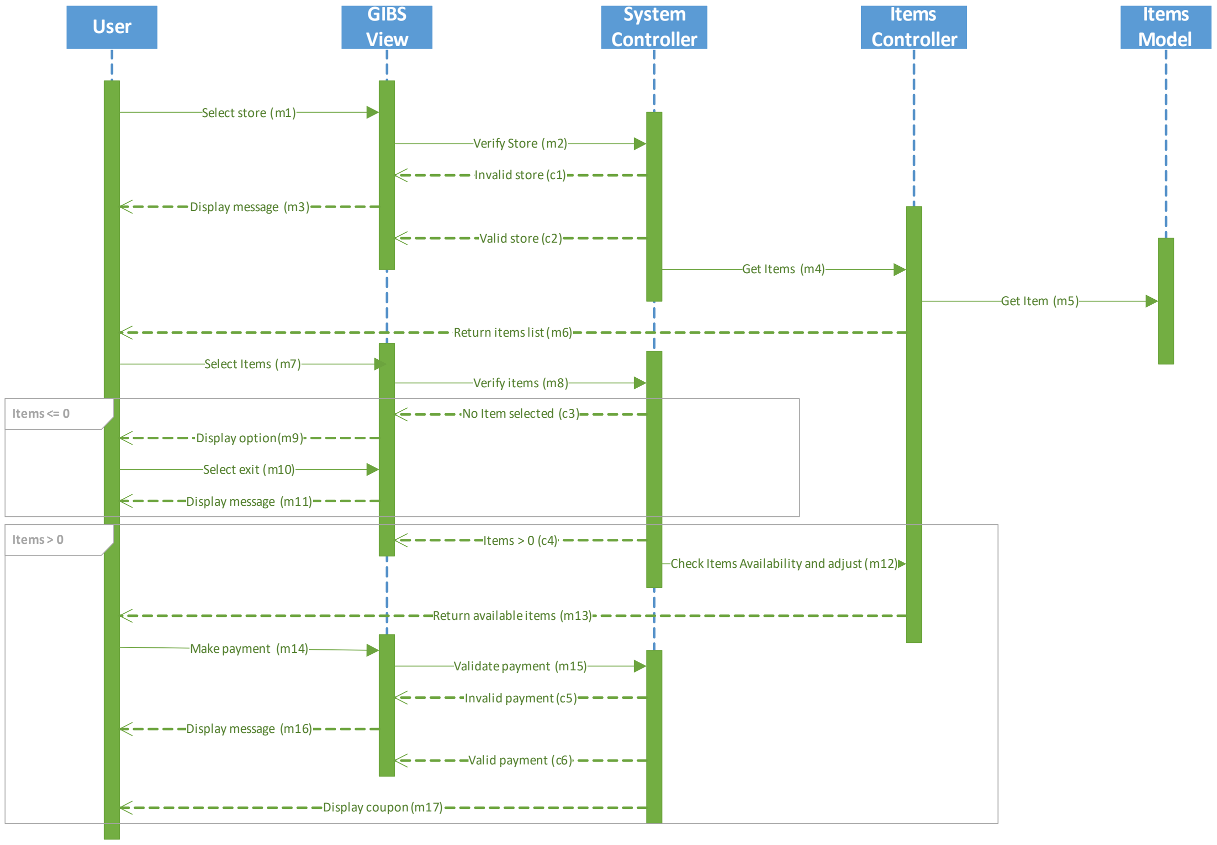

Figure 9: Sequence Diagram for GIBS Case Study

Table 6: Operation Scenario for GIBS

<Scn 1	<Scn 2	<Scn 3	<Scn4
State X	State X	State X	State X
S1: (m1, a, b)	S1: (m1, a, b)	S1: (m1, a, b)	S1: (m1, a, b)
S2: (m2, b, c) | c1	S2: (m2, b, c) | c1	S2: (m2, b, c) | c1	S2: (m2, b, c) | c1
S3: (m3, b, a)	S4: (m4, c, d)	S4: (m4, c, d)	S4: (m4, c, d)
State Y>	S5: (m5, d, e)	S5: (m5, d, e)	S5: (m5, d, e)
	S6: (m6, d, a)	S6: (m6, d, a)	S6: (m6, d, a)
	S7: (m7, a, b)	S7: (m7, a, b)	S7: (m7, a, b)
	S8: (m8, b, c) | c3	S8: (m8, b, c) | c4	S8: (m8, b, c) | c4
	S9: (m9, b, a)	S12: (m12, c, d)	S12: (m12, c, d)
	S10: (m10, a, b)	S13: (m13, d, a)	S13: (m13, d, a)
	S11: (m11, b, a)	S14: (m14, a, b)	S14: (m14, a, b)
	State Y>	S15: (m15, b, c) | c5	S15: (m15, b, c) | c6
		S16: (m16, b, a)	S17: (m17, c, a)
		State Z>	State Z>

The sequence diagram graph (SDG) can be created using Table 6 and is presented in Figure 10.

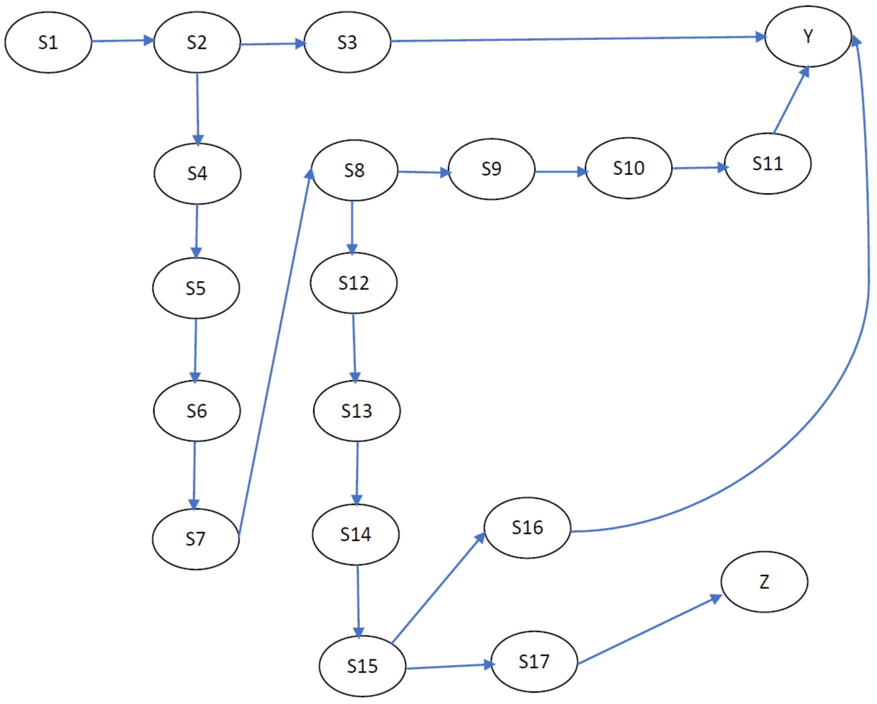

Figure 10: Sequence Diagram Graph (SDG) for GIBS

From Figure 10, the test case paths that are generated are listed below.

Path 1: S1-S2-S3 = Unsuccessful

Path 2: S1-S2-S4-S5-S6-S7-S8-S9-S10-S11 = Unsuccessful

Path 3: S1-S2-S4-S5-S6-S7-S8-S12-S13-S14-S15-S16 = Successful

Path 4: S1-S2-S4-S5-S6-S7-S8-S12-S12-S14-S15-S17 = Successful

Using above test case paths and by gathering more information, test cases are presented in Table 7.

From Table 7 we could infer that sequence diagram generate four test cases for GIBS case study. This is because sequence diagram does not cover all the looping or iteration activities of the case study. The sequence diagram failed to cover the looping activity when the selected store is invalid. It also failed to cover the looping activity when the payment is invalid. These are the two test cases which sequence diagram failed to cover. Because of this reason, it generates four test cases whereas activity diagram technique generates six test cases.

Table 7: Test Cases for GIBS Case Study Using Sequence Diagram

Test case scenario	Input	Output
1	Store = "Invalid"	Message would be to select a store.
2	Store = "Valid" Selected items = 0 Select Exit	Displays the initial screen with stores.

3	Store = "Valid"
	Selected items > 0
	All items are available
	Payment = "Valid"
		Displays the coupon

4	Store = "Valid"
	Selected items > 0
	All items are not available
	Payment = "Invalid"
		Displays the coupon

3.2.3 Test Case Generation Using State Chart Diagram

In this section we discuss technique proposed by [17] which uses state chart diagram to generate test cases. They have proposed a test case coverage criterion which would be useful when UML state chart diagram is used to generate test cases. It uses ATMW case study to demonstrate the coverage criterion. State chart diagram is useful in describing the behavior of a system and model dynamic nature of the system [1]. In this section, we discuss the coverage criterion presented in [17] technique using Grocery Item Buying System (GIBS) case study.

Figure 11 presents the state chart diagram for GIBS case study. The state chart diagram presents different states involved and the transitions between the states.

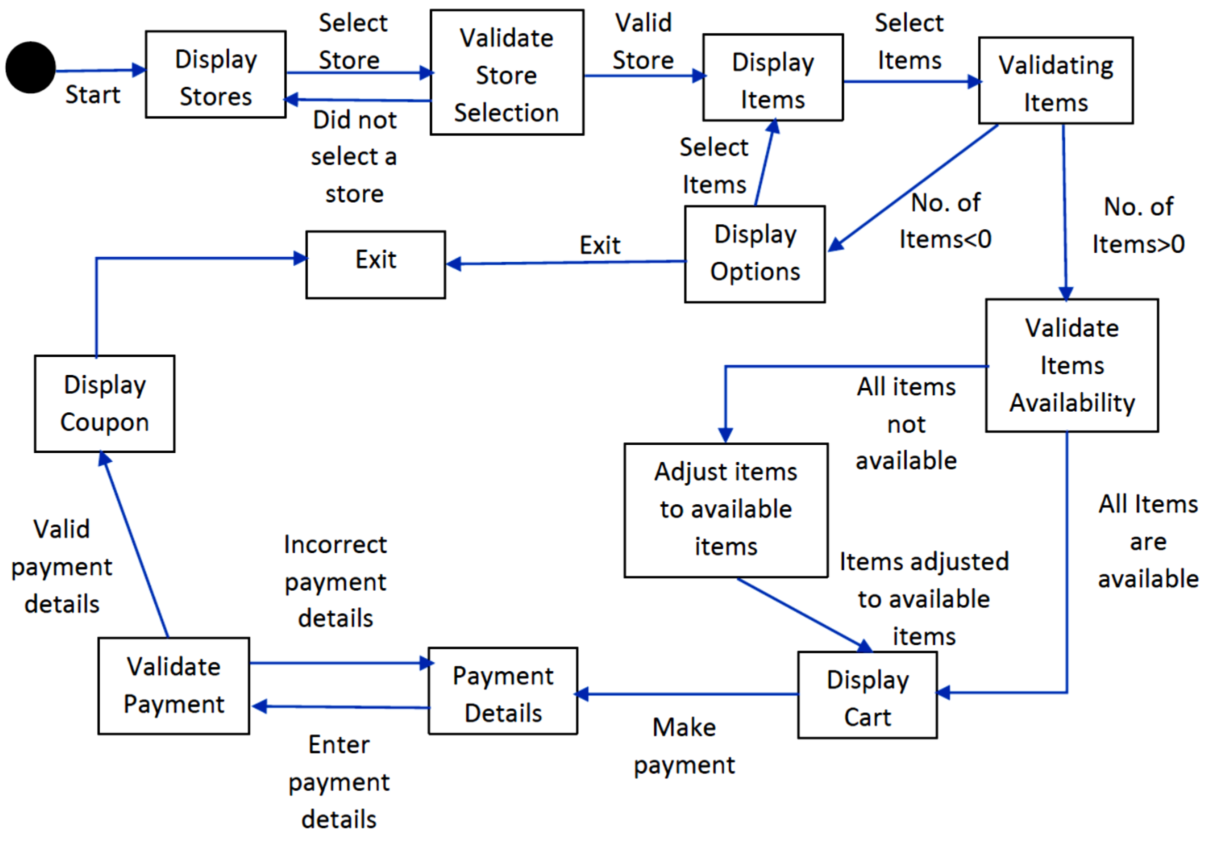

Figure 11: State Chart Diagram for GIBS Case Study

3.2.3.1 Technique Overview

In this technique, state chart diagram is converted in to state transition graph (STG). All the information that is required to generate test cases is extracted from STG. Using STG, we define the test paths and extract the test cases based on different coverage criterion like all-state coverage, all-transition coverage, all-transition-pair coverage.

3.2.3.2 Conversion of State Chart Diagram to State Transition Graph

State transition graph (STG) describes the process of the system using the vertices. STG is generated from state chart diagram by representing the state of a system using a vertex (V). STG contains a list of vertices which can be defined as Vi = (V1, V2, V3….) and there is a unique edge which joins two vertices. The edges represent transition between the states or vertices. Figure 12 presents the STG for GIBS case study.

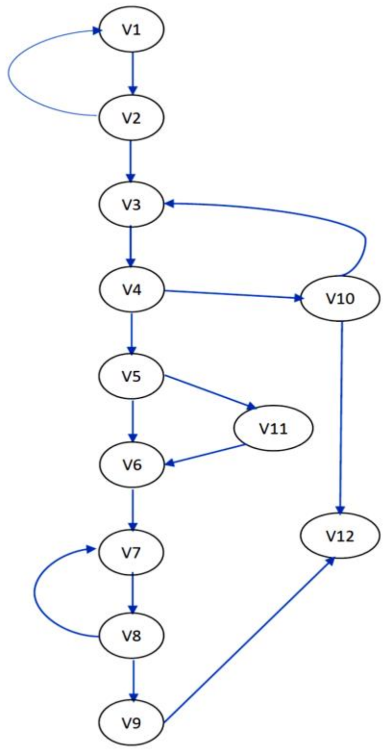

Figure 12: State Transition Graph (STG) for GIBS Case Study

3.2.3.3 Coverage Criterion

[17] discusses about different coverage criterion which would be useful in covering all the possible test paths. The different coverage criterion to consider while generating test case using state chart diagram are:

All-State Coverage specifies that each state should be visited at least once to satisfy this criterion. It is useful in covering all the states in a coverage criterion. This criterion helps in analyzing if all the states are visited and considered while generating test cases.

All-Transition Coverage specifies that each transition must be visited at least once to satisfy this criterion. It is useful in covering all the transition between the state and helps in analyzing if all the transition has been considered while generating test cases.

All-Transition-Pairs Coverage specifies that the test cases should considers all adjacent transition successively entering and leaving a state. It is useful in covering all the transitions leaving from a given state.

3.2.3.4 Generation of Test Cases

To generate test cases we consider state transition graph (STG) presented in Figure 6. From STG, we define the set of vertices as Vi = (V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11) and edges as Ei = (E1, E2, E3, E4, E5, E6, E7, E8, E9, E10, E11, E12, E13, E14)

The six test paths that are generated covering all the coverage criteria are:

Test Path 1: V1-V2-V3-V4-V5-V6-V7-V8-V9-V12

Test Path 2: V1-V2-V3-V4-V5-V11-V6-V7-V8-V9-V12

Test Path 3: V1-V2-V1-V2-V3-V4-V5-V6-V7-V8-V9

Test Path 4: V1-V2-V3-V4-V5-V6-V7-V8-V7-V8-V9-V12

Test Path 5: V1-V2-V3-V4-V10-V3-V4-V5-V6-V7-V8-V9-V12

Test Path 6: V1-V2-V3-V4-V10-V4-V10-V12

The GIBS case study generates six test cases when state chart diagram is used. The same case study generates four test cases when sequence diagram is used but when activity diagram is used GIBS case study produces six test cases. This inconsistency of test case between state chart or activity diagram with sequence diagram is because the GIBS case study involves looping activities. As sequence diagram fails to capture the looping activity of a use case, it does not cover all the test cases. Whereas, activity and state chart diagram can be individually chosen to generate test cases for case studies involving looping activity.

Chapter 4: Observations and Guideline

In this section, we will be discussing about the observation that we have made using the three techniques we discussed in the previous section. Based on the observations, we would discuss a guideline to eliminate the limitations of UML diagrams.

4.1 Observations

Based on the case studies and techniques applied in the previous section, we could infer that only GIR case study produces consistent results among two techniques that were used to generate test cases. The two test cases are demonstrated with different techniques to generate test cases, but all the techniques produce the same number of test cases in case of GIR case study. This is depicted in the Table 8 which shows the comparison of the test cases generated for GIR case study with different techniques of generation.

Table 8: Comparison of GIR Test Cases for Different Techniques

GIR test cases	Activity Diagram	Sequence Diagram	State Chart Diagram
Test case 1	Yes	Yes	Yes
Test case 2	Yes	Yes	Yes
Test case 3	Yes	Yes	Yes

The reason GIR case study generates three test cases even when different techniques are used to generate test cases is that the GIR case study does not have any looping or iteration activity. For this reason, all the three UML diagrams can be used to generate test case for the case studies that does not involve looping or iteration activity.

GIBS generates inconsistent test cases when different techniques are used to generate test cases. The reason is because the GIBS case study involves looping or iteration. Due to this, a sequence diagram alone cannot cover all the test case paths. Furthermore, normally modeled sequences are not complete and does not offer any information on when the model behavior will occur [18]. From the case study, GIBS which involves looping or iteration activity, we could infer that sequence diagram alone is not enough to cover all the test case paths for the case studies involving looping or iteration activity [1].

Table 9 displays the types of test cases generated for GIBS case study, from which we could infer that the sequence diagram technique does not generate the two test cases, whereas the other two techniques generate six test cases.

Table 9: Comparison of GIBS Test Cases for Different Techniques

GIBS test cases	Activity Diagram	Sequence Diagram	State Chart Diagram
Test case 1	Yes	Yes	Yes
Test case 2	Yes	Yes	Yes
Test case 3	Yes	Yes	Yes
Test case 4	Yes	Yes	Yes
Test case 5	Yes	No	Yes
Test case 6	Yes	No	Yes

From the above case studies and results which are presented in [1], we could infer that not all UML diagrams support looping or iteration activities. In the two case studies, GIR case study is the only one which produces consistent results using activity, sequence and state chart diagram. Whereas, the other case study generates inconsistent test cases when sequence diagram is used to generate test cases. The final number of test cases generated using different techniques are presented in the Table 10. Using these observation, a guideline is proposed in [1], which discusses about the UML diagrams that could be used in a specific scenario to generate consistent test cases. In the next section, we discuss the guideline proposed in [1].

Table 10: Test Cases Generated for a Case Study Using Different Techniques

Case studies\Type of UML diagram	**Activity Diagram**	**Sequence Diagram**	**State chart Diagram**
GIR	3	3	3
GIBS	6	4	6

4.2 Guideline

The guideline presented in [1] helps the readers and researchers to choose a correct set of UML diagram to produce consistent test cases. This guideline takes in to consideration the features of the case study and analyzes the best possible UML diagrams that could be used to generate consistent number of test cases.

This guideline is based on two factors: one of them is the features of the case study and the other is the type of UML diagram [1]. The features of the case study correspond to the looping or iteration activity of the case study or the concurrent execution of the case study. And the type of UML diagram depends on the features of the case study which could generate consistent test cases. There are certain rules that this guideline imposes which are proposed in [1]:

a) Sequence diagram is good at capturing the interaction between the components in the system, but it fails in capturing the looping and iteration activities in the system. Because of this reason, sequence diagram should never be used in a case study which involves looping or iteration activity.

b) Activity diagram which is one of the most commonly used UML diagram to generate test cases, describes about the sequence of operations or steps need to complete a use case. Activity diagram could be used to generate test cases for case studies involving looping or iteration activity, and concurrent activity. It could also be used for case studies where there is no looping or iteration activity.

c) State chart diagram presents the dynamic behavior of the system. The transitions occur due to an event on the system. State chart diagram could be used in all the case studies that the activity diagram could be used in.

d) For the case studies which does not involve looping or iteration activity, either of the three UML diagrams (activity diagram, sequence diagram, state chart diagram) could be used. They all generate consistent test cases when there is no looping or iteration activity.

e) For the case studies involving looping and iteration activity, activity and state chart diagram could be used alone to generate test cases. If you want to use sequence diagram to generate test case, you could use this along with Labeled Transition System (LTS). This combination of sequence diagram and LTS generate the same number of test cases as the activity diagram, and state chart diagram.

Table 11 summarizes the rules proposed by [1].

Table 11: Guidelines to Generate Consistent Test Cases

Features of the case study	**UML Diagrams**
Case studies which does not involve looping or iteration process	1. Activity Diagram 2. State Chart Diagram 3. Sequence Diagram
Case studies which involves looping process	1. Activity Diagram 2. State Chart Diagram 3. Sequence Diagram with Labeled Transition System (LTS) extension.

From Table 11 we could infer that if sequence diagram is needed to generate test cases for case studies which involve looping or iteration activities, it could be combined with LTS which helps in generating consistent test cases when compare with activity diagram and state chart diagram. LTS technique transforms the sequence diagram and captures all the actions that occur and change the state of the system [1]. LTS captures all the interaction and labels them. This would help in deriving the final set of test case paths.

4.2.1 Application of Guideline

This section presents the methodology discussed in [1] to generate consistent number of test cases by using sequence diagram in combination with LTS. We discuss about the methodology by considering GIBS as the case study. In the above section, we have discussed that sequence diagram alone is not enough to generate test cases in case studies which involve looping or iteration activity. To solve this problem and to generate consistent number of test cases using sequence diagram, a technique has been proposed by researcher in [19] which uses LTS technique along with sequence diagram. This technique generates the same number of test cases as activity diagram, and state chart diagram.

4.2.1.1 Creation of Sequence Diagram from GIBS Case Study

Figure 13 represents the sequence diagram for GIBS case study. It represents the interaction between user and the system. The user communicates with GIBS system and the messages or type of operation that are performed by the user and the system are depicted in the sequence diagram. Once the sequence diagram is generated for the GIBS case study, we could use LTS, which generates the functional feature behavior, from which we could generate the test case paths.

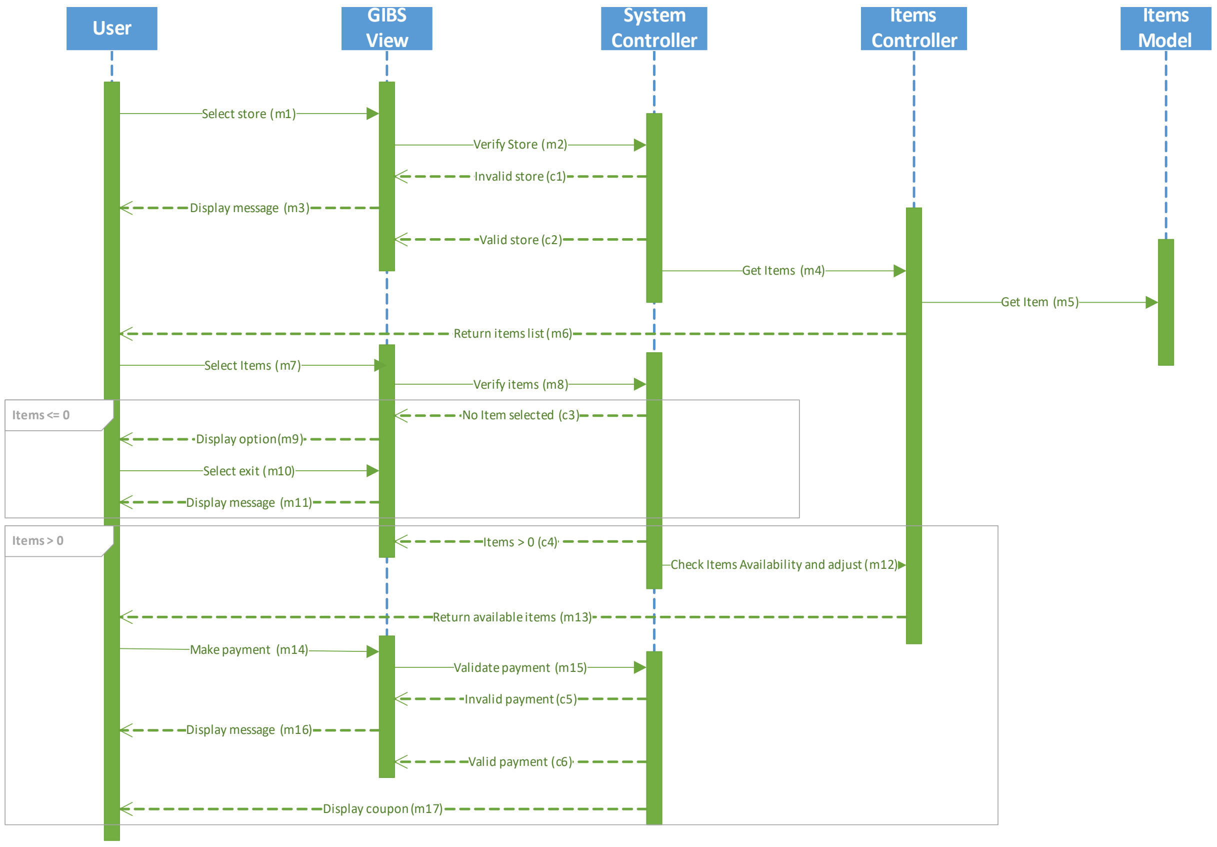

Figure 13: Sequence Diagram for GIBS Case Study

4.2.1.2 Transformation of Sequence Diagram in to Labeled Transition System (LTS)

LTS is one of the formal methods that can be used to represent the behavior of a system [20]. One of the problem with sequence diagram is its inability to represent looping or iteration operation. Because of this reason it is difficult to extract test cases covering these scenarios. This problem can be eliminated by using sequence diagram in combination with Labeled Transition System (LTS).

The researcher in [19] defined LTS as “a highly testable model that provides a universal, monolithic description of the set of all possible behaviors of the system”. Figure 14 shows the three elements of the system. (a) the initial state of the system, (b) the action performed on the system that initiates the transition of the state, (c) the final state of the system after the transition.

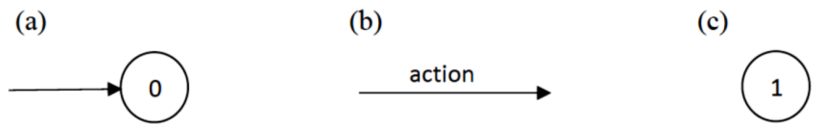

Figure 14: Elements of Labeled Transition System (LTS)

[1] defines LTS as a 4-tuple, S = (Q, A, T, q0), where:

Q is a finite, nonempty set of states,

A is a finite, nonempty set of states,

T is a transition relation, is a subset of QxAxQ,

q0 is the initial state.

To transform the sequence diagram in to LTS, the scenarios need to be obtained from the sequence diagram which represents the user actions and the system response. To add more details to LTS, annotations could be added which are steps, expectedResults, and conditions. The purpose of each annotation as defined in [1] is:

- Steps, the users action to be performed
- Expected Results, the systems response for user's actions.
- Conditions, a true of false situation which validates user's actions.

Figure 15 shows LTS which is created by transforming sequence diagram. LTS has nodes and transitions, with "0" being the initial node and incrementing the node count as new actions or transitions are performed. For each transition a new state is created.

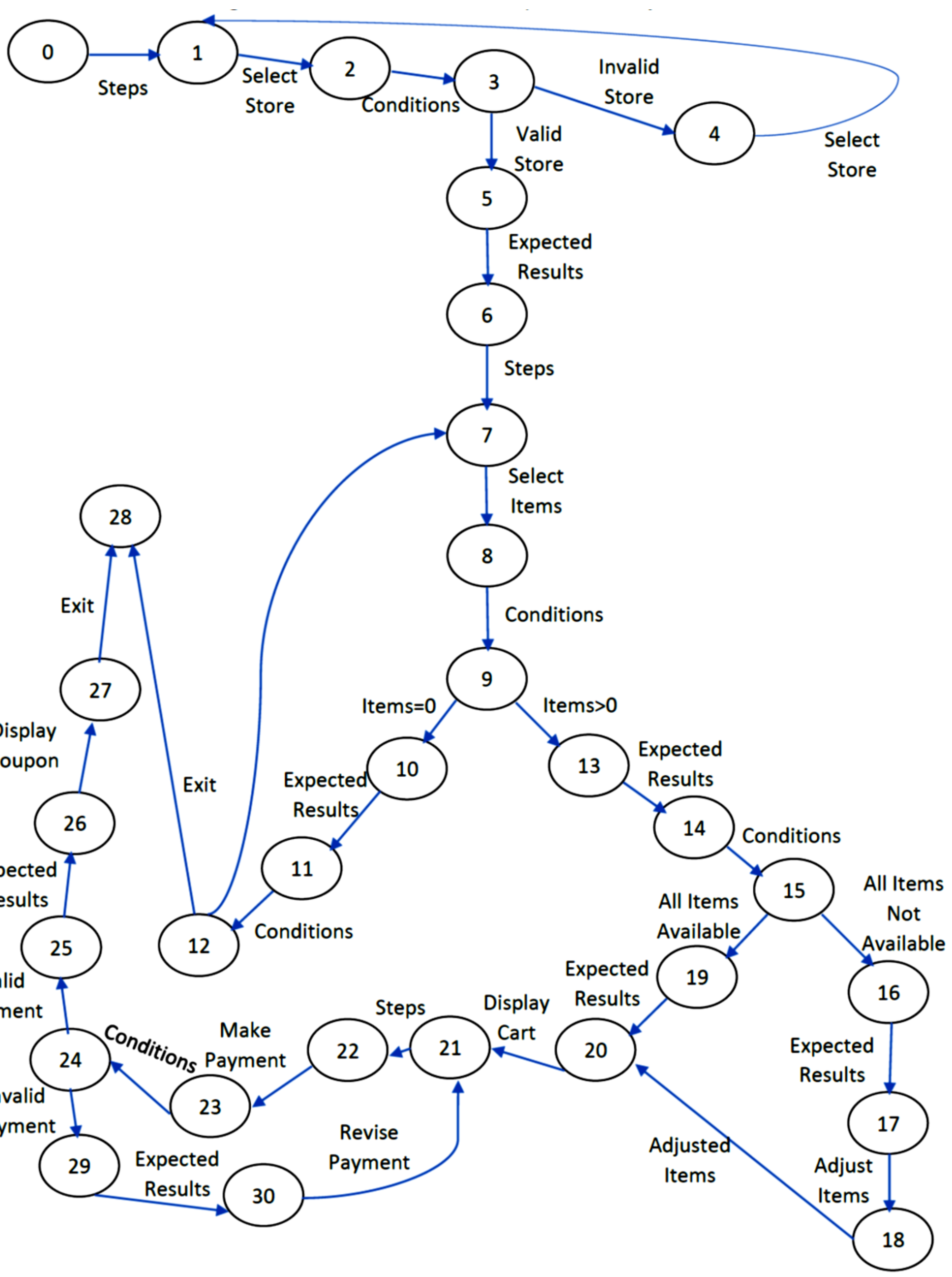

Figure 15: Labeled Transition System (LTS) for GIBS

The label "Steps" is used for transition from user-system or user-user. Whereas, "Expected Results" label is used to represent the transition from system to user. The label "Conditions" represent a state where the system validates user selection.

4.2.1.3 Generation of Test Cases

To generate test cases from LTS we use Depth First Search (DFS) technique. It helps in identifying all the paths from LTS. Using DFS, first the left branch is explored deeply before ntering the right branch. We start the DFS technique from the initial node of LTS. The generated test paths using DFS on LTS for GIBS case study are shown in Table 12.

The test paths are represented in the table and hence it is called Path Table. The Path Table shows that six test case paths are generated for GIBS case study. Each row in the path table represents a test case generated. Sequence diagram with LTS generates six test cases. test cases generated using this technique are consistent with the test case generated from activity diagram, and state chart diagram.

This method of generating test cases using sequence diagram and LTS extension is different from other techniques [1]. Other techniques use a UML model and covert in to a graph and generate test case from them. But this technique generates test cases by using an extension of LTS which is generated by transforming the sequence diagram. By using this extension of LTS, we eliminated the disadvantage of using sequence diagram to generate test case for case studies involving looping or iteration activity.

Table 12: Test Cases Generated for GIBS Case Study Using Sequence Diagram and LTS

Test case number	**Test case path**
1	Step: Select Store Condition: Invalid Store Expected Result: Display message and select store
2	Step: Select Store Condition: Valid Store Expected Result: Display Items Step: Select Items Condition: Item = 0 Expected Result: Display option Step: Select exit Expected Result: Exit
3	Step: Select Store Condition: Valid Store Expected Result: Display Items Step: Select Items Condition: Item = 0 Expected Result: Display option Step: Select Items
4	Step: Select Store Condition: Valid Store Expected Result: Display items Step: Select Items Condition: Item > 0 Condition: All items available Expected Result: Display cart Step: Make payment Condition: Invalid payment Expected Result: Revise payment
5	Step: Select Store Condition: Valid Store Expected Result: Display items Step: Select Items Condition: Item > 0 Condition: All items available Expected Result: Display cart Step: Make payment Condition: Valid payment Expected Result: Display coupon
6	Step: Select Store Condition: Valid Store Expected Result: Display items Step: Select Items Condition: Item > 0 Condition: All items not available Expected Result: Adjust items Step: Make payment

Sequence diagram with LTS extension generates consistent test cases compared with activity diagram and state chart diagram. As we have already observed that sequence diagram alone cannot be used to generate test cases for case studies involving looping or iteration activity instead we could use activity diagram, state chart diagram, or sequence diagram with LTS extension.

Chapter 5: Conclusion and Future Study

Generating test cases is a challenging task and requires a lot of effort. Generating and analyzing the test cases early in the software development process helps in detecting the faults earlier. Using UML diagrams to generate test cases is challenging as there are some advantages and disadvantages for each of them.

In this paper we have discussed case studies which have distinctive features. We have discussed about the techniques using different UML diagrams and discussed which would be the best one to choose in a case study based on the features. For the case studies which does not involve looping or iteration activity, we could use activity, sequence or state chart diagram. All the three UML diagrams produce consistent results for the case studies which does not have looping or iteration activity.

For the case studies which involve looping or iteration activity, sequence diagram alone might not be sufficient to cover all the test case paths. To achieve consistent results for such case studies we could either choose activity diagram, state chart diagram or a sequence diagram with an extension of Labeled Transition System (LTS). By using LTS as an extension we could use sequence diagram to generate the test cases which would produce consistent results when compared to test cases generated using activity or state chart diagram.

In this paper, we have discussed how the features of a case study affects in choosing a model to generate test cases. We have specifically looked at the looping activity in the use case and how it would affect the usage of models. Similarly, there could be discussion in future regarding the other features like nested looping, conditional statements that would impact the models. Discussing these would help software teams to better choose a test case generation technique based on the features of the case study. Based on the findings of the discussions the researchers could also find a solution to overcome the disadvantages of models.

References

[1] R. Abdul, N. A. J. Dayang, N. A. Ahmad, and R. Abdullah, “Selecting UML models for generation of test cases: An experiments of technique to generate test cases,” 2017, http://serl.fc.utm.my/index.php/ijset/article/view/86.

[2] S. M. Mohi-Aldeen, S. Deris, and R. Mohamad, “Systematic mapping study in automatic test case generation,” in *New Trends in Software Methodologies, Tools and Techniques*, H. Fujita et al., eds. IOS Press, 2014, 703-720.

[3] A. Kanjilal, S. Sengupta, and S. Bhattacharya, “Scenario path identification for distributed systems: A graph based approach,” in *Proceedings of 2010 21st IEEE International Symposium on Rapid System Prototyping,* 2010, pp. 1-8. 10.1109/RSP.2010.5656332

[4] K. Debasish and S. Debasis, “A novel approach to generate test cases from UML activity diagrams,” 2009, http://www.jot.fm/issues/issue_2009_05/article1.pdf.

[5] M. Utting, A. Pretschner, and B. Legeard, “A taxonomy of model-based testing approaches*,” Software Testing, Verification & Reliability*, vol. 22, no. 5, pp. 297-312, 2012.

[6] M. Mohamed, O. Samir, A. S. Waseem, and H. Abdelwahab, (2009). “A survey of model-driven testing techniques,” *2009 Ninth International Conference on Quality Software,* 2009, pp. 167-172. 10.1109/QSIC.2009.30

[7] J. Offutt and A. Abdurazik, “Generating tests from UML specifications,” in *The Unified Modeling Language, Lecture Notes in Computer Science,* vol. 1723. R. France and B. Rumpe, eds. Springer, Berlin, Heidelberg, 1999, pp. 416-429.

[8] A. Varikuti and D. Puvvula, “A novel approach for an early test case generation using genetic algorithm and dominance concept based on use cases,” *International Journal of Computer Science and Information Technologies*, vol. 3, no. 3, pp. 4218-4224, 2012.

[9] S. Asthana, S. Tripathi, and S. K. Singh, “A novel approach to generate test cases using class and sequence diagrams,” in *Contemporary Computing. IC3 2010. Communications in Computer and Information Science*, vol. 95, S. Ranka et al. (eds). Springer, Berlin, Heidelberg, 2010.

[10] D. Xu, N. Philbert, Z. Liu, and W. Liu, “Towards formalizing UML activity diagrams I CSP,” in *2008 International Symposium on Computer Science and Computational Technology, ISCST 2008,* vol. 2, pp. 450-453, 2012.

[11] E. Andrade, P. Maciel, G. Callou, and B. Nogueira, “A methodology for mapping SysML activity diagram to time petri net for requirement validation of embedded real-time systems with energy constraints,” in *Third International Conference on Digital Society ICDS*, 2009, pp 266-271.

[12] M. Sarma and R. Mall, “Automatic test case generation from UML models,” *10th International Conference on Information Technology (ICIT 2007),* pp. 196-201. 2007.

[13] N. Khurana and R. S. Chillar, “Test case generation and optimization using UML models and genetic algorithm,” *Procedia Computer Science*, vol. 57, pp. 996-1004, 2015.

[14] A. Tripathy and A. Mitra, “Test case generation using activity diagram and sequence diagram,” in *Proceedings of International Conference on Advances in Computing,* 2012, pp. 121-129.

[15] M. AzaharuddinAli, K. Shaik, and S. Kumar, “Test case generation using UML state diagram and OCL expression,” *International Journal of Computer Applications*, vol. 95, no. 12, pp. 7-11, June 2014.

[16] P. N. Boghdady, N. Badr, M. Hashem, and M. Tolba, “A proposed test case generation technique based on activity diagrams,” *International Journal of Engineering and Technology,* vol. 11, no. 3, pp. 37-57, 2011.

[17] Y. D. Salman, N. L. Hashim, M. M. Rejab, R. Romli, and H. Mohd, “Coverage criteria for test case generation using UML state chart diagram–1.5005458,” 2017, https://aip.scitation.org/doi/pdf/10.1063/1.5005458.

[18] D. Sokenou, “Generating test sequences from UML sequence diagrams and state diagrams,” *GI Jahrestagung*, no. 2, pp. 236-240, 2006.

[19] G. E. Cartaxo, G. O. F. Neto, and D. L. P. Machado, “Test case generation by means of UML sequence diagrams and labeled transition systems,” in *2007 IEEE International Conference on Systems, Man and Cybernetics*, 2007, pp. 1292-1297. 10.1109/ICSMC.2007.4414060

[20] N. F. As’Sahra and F. Komputeran, “Test case prioritization technique using sequence diagram and labeled transition systems in regression testing,” Universiti Teknologi Malaysia, 2015, https://books.google.com/books?id=isNPAQAACAAJ.