

*Thesis submitted in partial fulfilment of the requirements*

*For the award of degree of*

**Master of Engineering**

**in**

**Information Security**

*Submitted By*

**Priya**

**(801333017)**

Under the supervision of:

**Ms. Ashima Singh**

Assistant Professor

CSED



**COMPUTER SCIENCE AND ENGINEERING DEPARTMENT**

**THAPAR UNIVERSITY**

**PATIALA – 147004**

**July 2015**

## CERTIFICATE

---

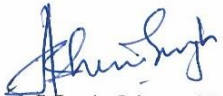
I hereby certify that the work which is being presented in the thesis entitled, “*Effective Test Case Generation in Regression Testing Using UML*”, in partial fulfilment of the requirements for the award of degree of Master of Engineering in *Information Security* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Ms. Ashima Singh* and refers other researcher’s work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

  
Signature:

**Priya**

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

  
**Ms. Ashima Singh**  
Assistant  
Professor  
CSED

Countersigned by  
  
**(Dr. Deepak Garg)**

Head  
Computer Science and Engineering Department  
Thapar University  
Patiala

  
**(Dr. S. S. Bhatia)**

Dean (Academic Affairs)  
Thapar University  
Patiala

## ACKNOWLEDGEMENT

---

First of all, I am thankful to God for his blessings and showing me the right direction. With His mercy, it has been made possible for me to reach so far, I wish to express my deep gratitude to Ms. Ashima Singh, Assistant Professor, and Computer Science & Engineering Department for providing her immense help, guidance, simulating suggestions and encouragement all the time. She always providing a motivating and enthusiastic atmosphere to work with; it was a great pleasure to do this thesis under her supervision.

I am also thankful to Dr. Deepak Garg, Head, Computer Science and Engineering Departments for his Kind help and cooperation. I express my gratitude to all the staff members of Computer Science and Engineering Departments for providing seminars and encouraging towards research work.

I want to express my appreciation to every person who contributes with either inspirational or actual work to this thesis. Last but not the least I am highly grateful to all my family members for their inspiration and ever encouraging moral support, which enables me to pleasure my studies.



Priya

## ABSTRACT

---

Component-based software system may contain external components as well as in-house built components. During the preservation phase the components get altered or modified very often. The type of testing which not only ensures that the modified component is working fine but also ensures that the changes have no adverse or effects on the rest of the system is called as regression testing. But due to lack of the knowledge about the third party mechanisms and the modifications done in the components, it is difficult for the component users or testers to do an appropriate selection of the test cases from the original test suite for testing the altered system. Thus there is a need of an efficient regression test selection approach which results in a reduced and effective regression test suite. The paper proposes an approach 'Regression Test Selection and Recommendation' which uses UML state chart diagrams and sequence diagrams to identify the changes which are further used for classification of initial test suite, selection and recommendation of test cases and development of a regression test suite. The aim is to reduce the size of the test suite effectively and also adhere to the quality of regression testing. The approach is validated using case study of **Automated Teller Machine**.

In recent years, Component-based software has been widely used in various application domains and becomes a fairly popular software form. However, Due to the lack of information about the externally developed components, system testers (i.e., component users) generally can't perform effective testing (especially regression testing) on their component based systems. Component users don't know the details about change in component, so they aren't able to select the proper test cases to retest the modified system. In this paper, we present an improved regression testing method based on the enhanced change information of component version to test the software system containing some modified components.

It is a collaborative testing method, needing the joint participations of component developer and user. Component developers calculate the change information from labelled method call graphs. Component users use this change information and their instrumentation records together to pick out test cases for next-round testing. In addition, we have employed preliminary experiments on some medium scale systems, the experiment results show that our regression testing method is fairly feasible and cost-effective in practice.

# TABLE OF CONTENTS

---

Certificate	i
Acknowledgement	ii
Abstract	iii-iv
Table of Contents	v
List of table	vi
List of figure	vii
List of Abbreviation	x
Chapter 1: Introduction	1
1.1 Component- Based Software Engineering	1-3
1.2 Software Component's Properties	4
1.3 Software Module Testing	4
1.4 Software Module Testability	7
1.5 Regression Testing of Component Based Software	10
1.6 Organization of Thesis	13
Chapter 2: Literature Review	14-25
Chapter 3: Problem Declaration	26
3.1 Research Gap Analysis	26
3.2 Problem Formulation	26
3.3 Objectives	27
Chapter 4: Proposed Methodology: Regression Test Selection and Recommendation	28
4.1 Overview	28-30
4.2 Detailed Description of Proposed Approach	30
4.2.1 Change Identification and Analysis	31-46
4.2.2 Development of Regression Test Suite	46-49
Chapter 5: Implementation and Experimental Results	50
5.1 Overview	50
5.2 Application of Proposed Approach RTSR	55-80
5.3 Results	84-86
Chapter 6: Conclusion and Future Scope	87
6.2 Contribution	87
6.3 Future Scope	88
References	89-92

## LIST OF TABLE

---

TABLE NO	TABLE DESCRIPTION	PAGE NO
TABLE 1	SUMMARY OF TABLE	22
TABLE 2	FINAL CHANGE IDENTIFICATION FOR SYSTEM A	46
TABLE 3	TEST SUITE FOR SYSTEM A	47
TABLE 4	REGRESSION TEST SUITE FOR SYSTEM A	49
TABLE 5	STATE CAHRT BASED CHANGES	61
TABLE 6	SEQUENCE BASED DIAGRAM CHANGES	70
TABLE 7	MAPPED SEQUENCE DIGRAM BASED CHANGES WITH STATE CHART BASED CHANGES	72
TABLE 8	FINAL SET OF CHANGES FOR ATM SYSTEM	72
TABLE9	ORIGINAL TEST SUITE FOR ATM SYSTEM	73-74
TABLE 10	CLASSIFICATION OF ORIGINAL TEST SUITE	85

## LIST OF FIGURE

---

<b>FIGURE NO.</b>	<b>FIGURE DISCRIPTION</b>	<b>PAGE NO</b>
FIGURE 1	Component Based Software Engineering Process Model	2
FIGURE 2	Software Component Testability Characteristics	7
FIGURE 3	Software Component Traces	9
FIGURE 4	Regression Testing Technique	11
FIGURE 5	Show the classification of test classification of test case prioritization technique	12
FIGURE 6	Regression Testing using enhanced changes Information	15
FIGURE 7	Generation of compatibility and prioritized Regression Test Suite	17
FIGURE 8	State Based Regression Testing	19
FIGURE 9	A Schematic of S-RTS Methodology	21
FIGURE 10	Cause- effect Diagram for inefficient Regression Testing	24
FIGURE 11	Regression Test Selection and Recommendation (RTSR) Approach	29
FIGURE 12	Addition of a State	31
FIGURE 13	Removal of a State	32
FIGURE 14	Addition of a Transition	32
FIGURE 15	Removal of a Transition	33
FIGURE 16	Changed Event	34
FIGURE 17	Changed Action & Change Guard Condition	35
FIGURE 18	Addition of Method	36
FIGURE 19	Removal of Method	37
FIGURE 20	Modified Method	38
FIGURE 21	Modified Sequence of Method	39

FIGURE 22	Addition of Object	40
FIGURE 23	Removal of Object	41
FIGURE 24	Addition of Option Combination Fragment	42
FIGURE 25	Changed Option Combination Fragment	43
FIGURE 26	Change in the State Chart Diagram for system A	44
FIGURE 27	Change in the Sequence Diagram for System A	45
FIGURE 28	Flow Chart Of ATM System	52
FIGURE 30	State Chart Diagram of Original ATM	56
FIGURE 31	State Chart Diagram of Modified ATM	58
FIGURE 32	Sequence Diagram of Valid Pin of ATM System	62
FIGURE 33	Sequence Diagram of Invalid Pin of Original ATM System	63
FIGURE 34	Sequence Diagram of Invalid Pin of Modified ATM System	64
FIGURE 35	Sequence Diagram of Cash Withdrawal from Savings of Original ATM System	66
FIGURE 36	Sequence Diagram of Cash Withdrawal from Savings of Modified ATM System	68
FIGURE 37	Home page	75
FIGURE 38	Deleted Transition	76
FIGURE 39	Obsolete Test Cases	77
FIGURE 40	Changed Transition	78
FIGURE 41	Retestable Test Cases	79
FIGURE 42	Reusable Test Cases	80
FIGURE 43	Add the Cases	83
FIGURE 44	Regression Test Suite	84
FIGURE 45	Classification of Original Test Suite	85
FIGURE 46	Percentage Reduction	86

## LIST OF ABBREVIATION

---

Abbreviation	Description
CBSE	Component Based Software Engineering
CBD	Component Based Development
RTS	Regression Test Selection
UML	Unified Modelling Language
RTSR	Regression Test Selection and Recommendation
ATM	Automated Teller Machine

### **1.1 Component Based Software Engineering:**

Component Based Software Engineering (CBSE) means the development of system by using already developed software component. Component based software system may contain external components as well as in-house built component. The components are stored in the repository which is accessible to programmers. Component based development approach builds the software system by choosing appropriate components and then integrating them by using a well-defined architecture.

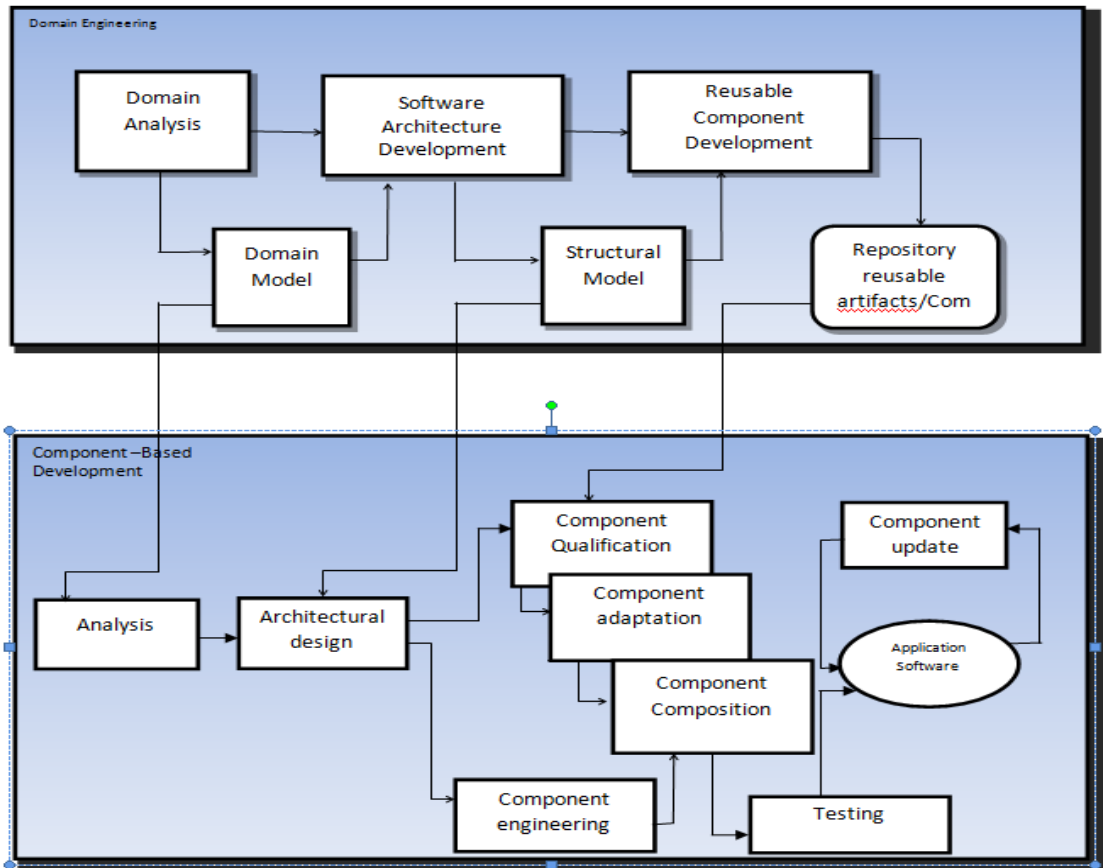
### **Need of Component Based Software Engineering**

Software systems have become more complicated which causes higher cost of building the systems, lesser productiveness, and lowering in the system quality. Thus need of finding a good economic development strategy arises. CBSE helps in decreasing the cost of building the systems, the time it takes to deliver to the customer, enhances the maintenances of the software and dependability on the system.

### **Component Based Software Engineering Process**

The Component Based Software Engineering Process involves recognizing candidate components, qualifying the interface of each components, adjusting components to eradicate architectural conflicts, assembling the components into a defined architecture and also updating components when there is need of changing the system.

Fig.1 shows the process model for CBSE, focusing the domain engineering goes on simultaneously with the component based development (CBD). Domain engineering constructs a domain model of applications which is used during CBD for analysis for user requirements, a structural model which is used as an input to architectural design, and provides reusable components to developers for component based development.



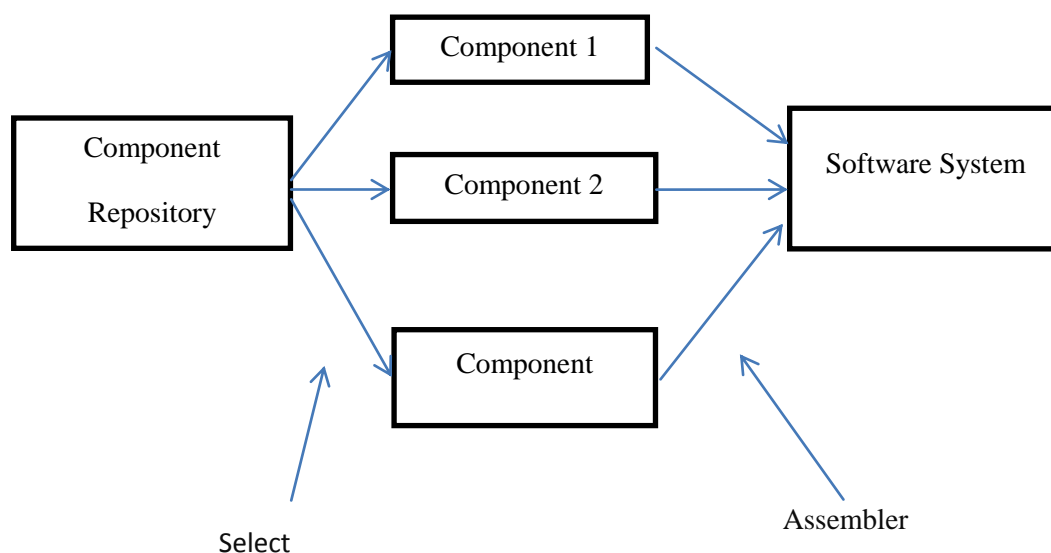
**Fig: 1: Component Based Software Engineering Process Model [1]**

## Domain Engineering

Domain engineering focuses on recognizing, constructing and publicize those components which can be used with the current and future software system in a specific domain. It involves analysis of the application domain during which the domain analyst finds out duplicate patterns in application inside a domain to design an application domain model, development of architecture for constructing structural model containing few numbers of structural elements showing unambiguous patterns of interaction and development of components which can be reused and stored in a repository which can be used by developers during components based developments.

### Component-Based Development

Components –Based Development (CBD) is an approach of developing software systems by using components. The programmer uses already existing components to fulfil the desired function which is required in the application. Let's say, if a components created by some developers which allow a user to “log in” then other programming can use it in their applications for that functionality. As in conventional concept software systems are developed from the beginning, CBD varies from the concept. Fig 2 describes that components are picked up and then integrated into the destined system.



Component development involves analysis of customer requirements and enhancements of architecture which is suitable in accordance with the analysis model being designed for the application. Then architecture is supplied with components which are either already existing or newly developed. If the components is available to be reused then the component qualification (i.e. ensuring that the selected components executes all the desired functions, appropriately adjust into the specific architecture and reveals the quality attributes such as availability, security, portability which are essential for the application), components adaptation (i.e. ensuring that constant ways for managing resources are used for every component, for every component a same set of activities are there and interface are realized in a constant way) takes place, however if not then component is engineered and then finally all the components are integrated and tested.

## **1.2 software component's properties**

Component is a portion of the system which is not dependent and can be replaced and does its functioning with respect to a clearly-determined architecture.

## **1.2 Properties of Software Component's**

### **Component**

Component is a part of the system which is not dependent and subjected to be replaced and did its job under specific architecture.

### **Properties of a software component in CBSE [3]**

1. **Personality:** Every components should be extra-ordinary indefinable in that surrounding in which it is used to be build up also in that surrounding in which it is to be deployed.
2. **Co-dependent delivery :** Components should be provided as uncommitted part to components users so that they can use them for building a components-based system
3. **Modularity and portability:** software systems are sub-divided into parts known as software components thus they possess some standard.

4. **Deployable:** A software module is known as deployable as and when it is build using a clearly stated approach. It will outcome in an executable instance in the atmosphere in which it is to be deployed.
5. **Recycling:** Every software module should have the property of being reused. The possessions **which** are reusable are analysis of specification, mean, foundation code, and executable etc.
6. **Customizability and casing:** This trait means that the software module can be customized and packaged to meet the most wanted functionality. Such components are called customizable components.
7. **Interoperability:** A module is having interoperability when it sanctions interactions and shift of data with other components. It can be classify as local and isolated interoperability. Local ans isolated interoperability means a horde centred atmosphere for components and isolated interoperability means components are on system.
8. **Symphony:** It gives access to the components to have symphony liaison which helps a component to fabricate and flatten other components. It is transitive if 1 comes under category II and category II comes under Part A then basically 1 comes under the Part A.
9. **Sculpt orthodoxy:** Module should be developed under a specified state sculpt of manner, edge styles etc.

### 1.3 Software Module Testing

There are already so many definitions of Software module testing have been made, here we discuss some of them such as

Module testing refers to testing, analysing the module individually or a group of interrelated modules [4]. This also may be further describing as a set of actions to find out errors and ensure the quality of individual's components [3].

Software module testing has certain characteristics which are same as that of software testing. Major 4 of them are briefly defined below:-

### **Software Module Testing Personality.**

- 1. Self-motivated:** Testing can be either dynamic or static, in dynamic testing actual implementation of system under test take place with certain inputs to check whether the output is as anticipated or not, by executing the test cases. While static testing takes place during early stages of software development life cycle i. e. Usually before the coding phase and this testing is done without actually running the system under test. It is ideal that testing should be dynamic to assemble the user's needs as it involves authentic running of the system and hence gives more obvious result as evaluate to static testing.
- 2. Determined:** The number of test cases can be infinite sometimes because a variety of combination of the condition to be tested, which makes testing infeasible in terms of time and cost. Therefore the total space should be infinite i.e. the number of test cases to be actually executed must be finite so that it takes less time and resources for testing.
- 3. Preferred:** Test cases should be properly selected from a vast accessible test collection according to specific testing criteria for test selection and treatment to achieve cost-efficient and adequate fault instructive testing.
- 4. Anticipated:** Testing should result in a decision that a test had succeed or failed in order to access anticipated reliability and quality of the software. For this purpose test oracle is used which generated desired testing result for certain input and also compares the actual testing result for definite input and also compared the actual testing result with the most wanted

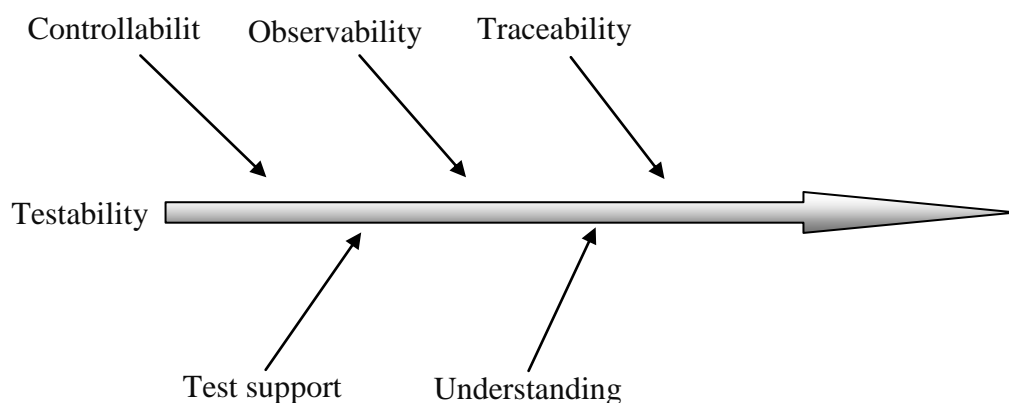
### **5. Result.**

## 1.4 Software Module Testability

Testability is basically means ability to be tested or in other words we can say testability means the scope to which software allows setting up of criteria for testing and efficiency of tests to make sure that if the well-known criteria is met or not or it refers to the degree to which the constraint are described in such a way that they allow setting up of a criteria for testing and efficiency of tests to check if the well-known criteria is met or not [4]

### Software Module Testability Personality

Software testability is considered as one of the major factors to compute software reliability. In order to make component reliability measurable or quantifiable, the following characteristics can be used for measuring software component testability effectively. Shows characteristics which can be used to solve the problem of measuring of testability.



**Fig: 3. Software component testability characteristics [7]**

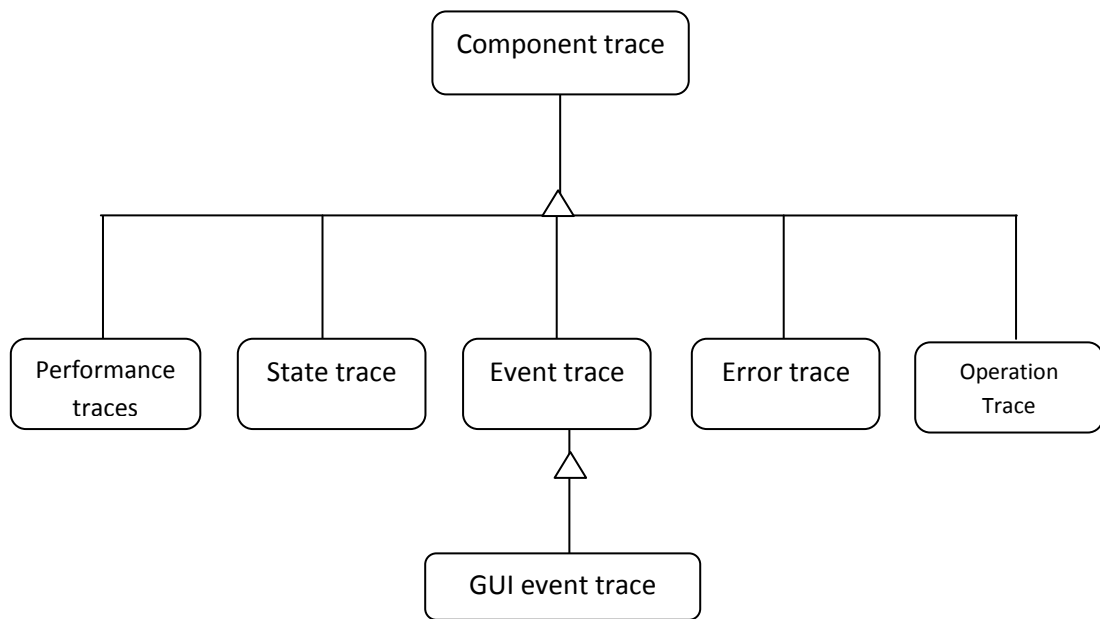
Freedman [8] considers two factors observability and controllability to illustrate testability of domain. Binder [9] also uses these as features of testability and traceability for representing testability and testing environment. Gao et al. [3] [10] express component testability by using following properties.

- 1) **Component Traceability:** Component traceability means the degree to which a component is able to track its functions, features and the way it behaves internally or externally.

A component which is traceable allows all its essential details for testing i.e. which are required to describe the way it execute, to be recorded. The major traces which can add value to testing of component are.

**Operation Traces** – Keeps a track of communication within a component and between different components.

- I. **Performance Traces:** keeps a track of the data related with efficiency and also of the standards for every method in a component for a specific work environment. These traces are also used to recognize problems related to performance testing
- II. **State Traces:-** Keeps a track of all the states in a component.
- III. **Event Traces-**Keeps a track of an event and sequence which has taken place in a component.
- IV. **Error Traces** – Keeps a track of information regarding errors, any exception and corresponding details being produced by component.



**Fig: 4: Software Component traces [3]**

**Component Observability:** Component observability means the extent to which it is simple to monitor the testing details on the basis of how a component behave, the given input, resultant output when executed for a test case. Clearly designed interface can improve the capability of a component to be observed, which allows us to identify a relation between input given and subsequent output for a test case during testing.

- 2) **Component understanding:** Component understand ability means extent to which it is simple to understand the details of a component, allowing the tester to simplify use the useful details such as requirement and specification to perform testing and also create efficient test for software component testing. The key factors associated with this property are firstly availability i.e. whether all the document like SRS, readme , design, code are available or not and secondly understand ability i.e. whether the details are represented in a presentable way such that it is easy to read and understand it or not.

- 3) **Component Test Support capability:** Component test support capability means ability of a component to generate tests, ability to organize tests, ability to evaluate and analyse test coverage and ability to execute tests and support testing.

## 1.5 Regression Testing of Component Based Software

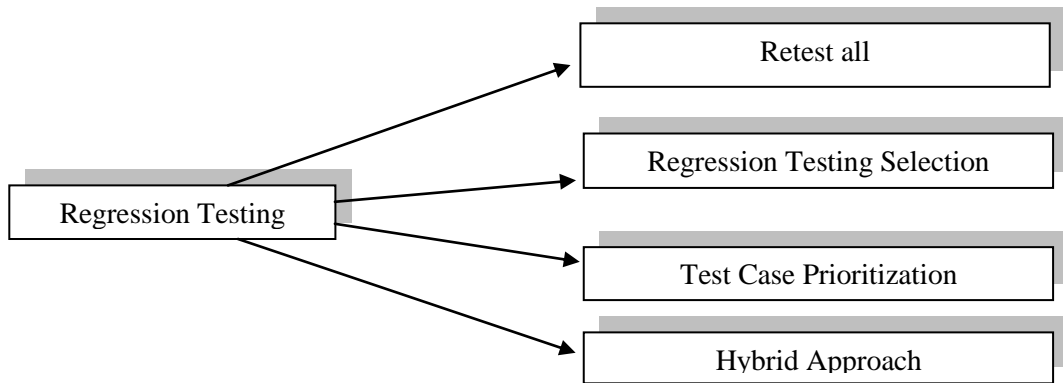
Regression testing means testing a system or a part of the system again to check that any change which has been done is implemented correctly and also the rest of the system is working fine as desired. It involves selecting some of test cases from existing ones and executing them again to test the changes and their impact on rest of the system. Regression testing is needed when some change has occurred in the original system as the new feature can dependent on some other existing feature thus it is a quality check to ensure that the existing feature are as same as they were earlier. A change can occur in a system as there may be change the requirement of the client which is very common in real scenario, or there can be a need to add some new functionality into the existing system another major reason can be whenever a bug or defect is fixed sometimes when we try to fix that bug may lead to some new defects being introduced into the system.

**Regression testing is applicable at all the levels.**

- i. **Unit Level:** Regression testing can be done at unit level in order to test the change by testing the individual unit being modified.
- ii. **Integration Level:** Let us say there is some change in two unit or components of the system, by doing regression testing on individual unit we can just test the that individuality the components are working fine but we must ensure that when integrated together then also they must as well as expected, hence once those parts are tested alone then they are integrated and tested again to ensure that the changes are implemented properly and don't have any bad impact on each other.

- iii. **System Level:** it is very necessary to do regression testing at the system level as the system is delivered to the client as a whole and not its individual parts are delivered. Hence if a change has occurred then once change is being tested then it is very important to ensure that it doesn't have any bad effect on the rest of the system which can be assured by doing regression testing of the whole system.

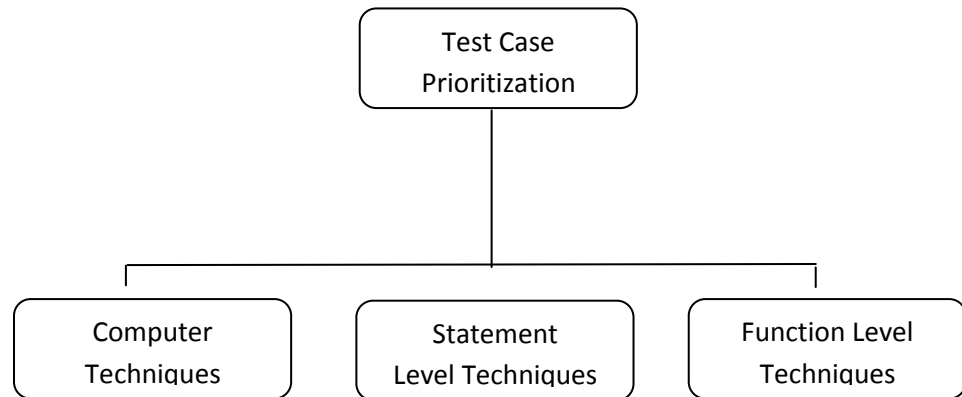
**Regression Testing Techniques:** Regression testing can be done by different techniques, which are shown below in the figure.



**Fig.5: Regression Testing Technique [11]**

1. **Retest All:** In retest all technique, the whole test suite is made to execute again to test the change and their impacts. It is very time consuming task and leads to increase in cost of testing.
2. **Regression Test Selection:** As retest all is a time consuming and costly method, therefore rather than we run the whole set of test cases again and again ,we select some of the test cases which correspond well with the changes and execute them to test the change and the system and this technique is called regression test selection.it can be further categorized as coverage techniques which uncover those elements in which there is some change and then close the corresponding test cases, minimization technique which picks out least amount of test cases and safe technique which choose the ones that generate some other outputs for the changed system in comparison with existing system.

3. **Test Case prioritization:** In these techniques the test cases are selected on the basis of priority, the test cases are prioritized by using some techniques which are categorized into three categories;



**Fig. 6 shows the classification of test case prioritization techniques.**

**Comparator Techniques:** The Selection of the test case is done through two methods- random ordering in which prioritization of test cases is done in an arbitrary manner and optimal ordering in which prioritization of test cases is done according to rate of error detection

**Statement Level Techniques:** The test cases can be selected using different methods- total statement coverage prioritization which prioritize a test case according to the count of statements, more the statement are covered higher the priority is given, another criteria is additional statement coverage prioritization which is alike the previous method but it also ensure that each line of code is checked by minimum 1 test case, another way is total fault exposing potential prioritization – the test cases are prioritized on bases of chances a test cases can uncover a fault and last way is additional fault exposing potential prioritization just extension of previous discussed.

- a. **Function Level Technique:** The test cases are selected as similar to statement level techniques with a difference that these consider function instead of statement

- 4: Hybrid Approach:** Hybrid technique is a combination of test selection technique and prioritization technique so as to attain the benefits of both the approach.

### **Organization of Thesis**

- I.** Chapter 1 discusses the basic concept related to CBSE and regression testing so as to have a basic knowledge about these Concepts
- II.** Chapter 2 gives the overview and analysis of various approaches being purposed for regression testing of component based software.
- III.** Chapter 3 states that research gap analysis, formulates the problem and objectives.
- IV.** Chapter 4 describes the purposed approach- “Regression test selection and Recommendation for Component based software”.
- V.** Chapter 5 validates the purposed technique using case study.
- VI.** Chapter 6 concludes the work done and stated the future scope of work.

## Chapter 2

### Literature Review

---

This section describes an overview of various approaches being proposed for regression testing which is applied on the changed software to ensure that the modified parts are working as expected and the existing parts are not affected by three changes done on the original software. The paper stated first safe regression test selection technique for software developed in java which efficiently handled the features of the language such as Polymorphism, dynamic binding, exception handling. Also a regression test selection system named RETEST [12] that used this technique was developed. The technique was efficiently able to reduce the size of test suite but the amount of reduction was different for different versions.

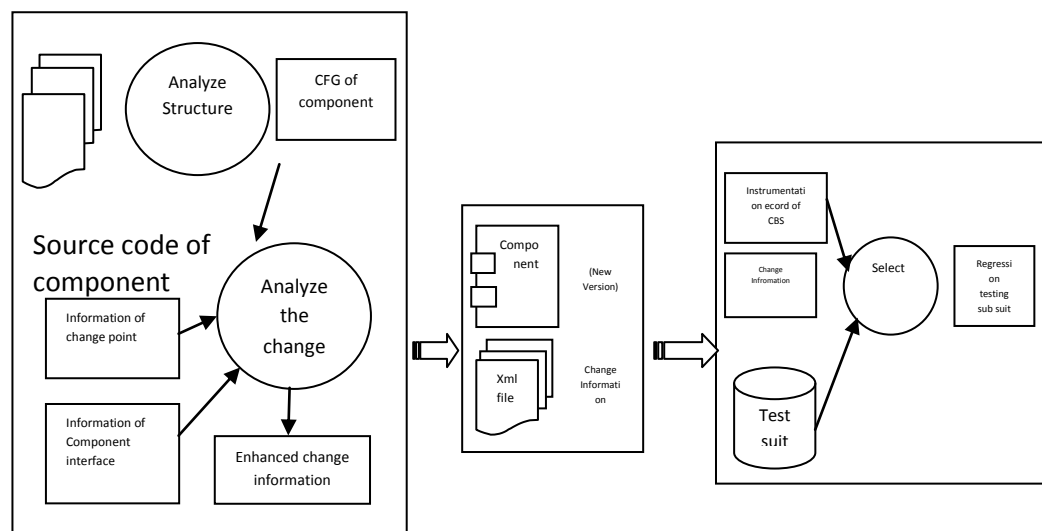
In the year 2001, Orso et al. [13] purposed two approaches for solving the problem of test cases for performing regression testing of component based software using the meta-content of the component. Code-based test selection techniques [14, 15, 16, 17] develop the graphical representation (like control flow graph) if the original program and then execute the test suite to find out the percent coverage with respect to some entity (like edges) in the graphical representation. Then the modified program is also represented in the same way and the two representation are compared to find out the difference between the two on the basis of entity being measured for coverage and then those test cases are selected which cover those entities being modified. The meta- contents required for selecting test cases for code based regression testing are test suite coverage percentage in context of edges, versions of the components and information about the modified edges. Specification-based technique construct test cases basis on functional specification of the system and find out the coverage of the test suite percentage in context with entity in the functional representation. Themeta-contents required for selecting test cases for specification based regression testing are test

suite coverage percentage in context with test frames [18], versions of the components and a way to find out information about the modified test frames. The approach is able to reduce twenty six percent of the effort required for testing but certain factor like the cost of making the meta-contents available and selecting the test cases are needed to be worked upon.

In the year 2005, Mao and Lu [19] purposed a technique for regression testing of component based by using improved information about the changes implemented in the original components. A new technique emerged involving both the developer of the component and the user of the component. The component developer provides the enhanced change information along with the component in form of a xml file which helps the user to select a subset of the existing test suite for performing

Regression testing. The technique is practical and also reduces the cost of testing for medium scale component based system but is yet to be validated for large and real time systems.

The fig. 7, below shows the complete process of constructing a regression test suite using enhanced changes information.

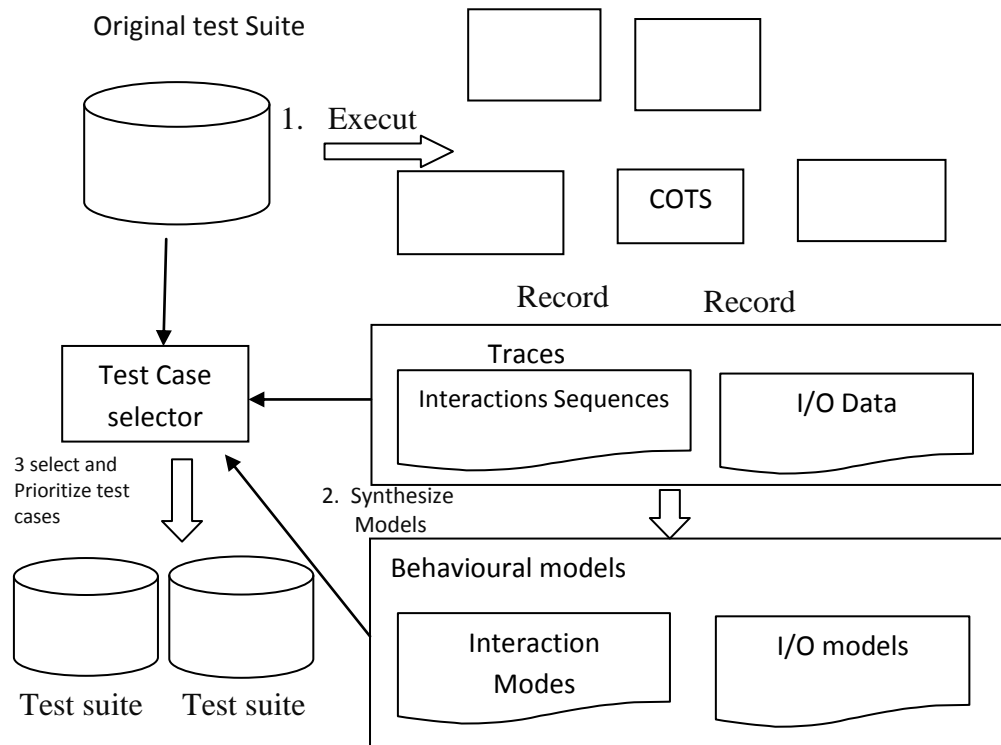


**Fig. 7: Regression testing using enhanced change information [19]**

In the year 2005, Muccini, Dias and Richardson [20] presented a paper enlightening the concept of doing regression testing at an architectural level. There are certain existing testing methods to verify that whether the actual behavior conformed to desired behavior [21] and to conduct integration testing [22, 23] the aim was to provide a framework and a method for regression testing when changes have been introduced to both the software architecture and also to the implementation. The paper focuses on two goals, firstly how to rescue the test cases at code level to ensure that changed code corresponds to specification at software architectural level and secondly how existing test cases defined at the architectural level can be reused to test the modified architecture.

In the year 2006, Gao et al. [24] presented a paper to address certain issue regarding regression testing like ways to make out the changes in a component, effect of those changes on the component, identifying the test cases which can be reused from the existing ones. Whenever a component goes some kind of change it is necessary to ensure that the component is working as per the desired requirement and then only it should be combined with must be tested and then it should be incorporated in the whole system. The paper proposed a retesting technique which deals with API based changes and effect on the component. A tool named COMP Test [24] based in this technique is developed which finds out API based changes in a component, impacts of those changes and the test cases which can be reused from the test suite of the component in an automatic manner.

In the year 2007, Mariani, Papagiannakis and Pezz'e [25] presented a paper which focused on two issues: firstly to rapidly recognize those components which conform to the interface specification but do not integrate well in the application environment and secondly to automatically produce the test case for regression testing. A technique was proposed for solving such problem with the help of compatibility testing and prioritized regression testing. Compatibility testing is intended to construct a test suite which is able to rapidly find out the remove components which are not compatible and prioritized regression testing is intended to construct a prioritized test suite which is small in size and which is able to detect as many faults as possible leading to reduction in testing cost and thus saves the time and effort needed for testing. Both the test suite are generated in an automatic manner from the behavioural models being produced by behavioural test and Capture technique [26].



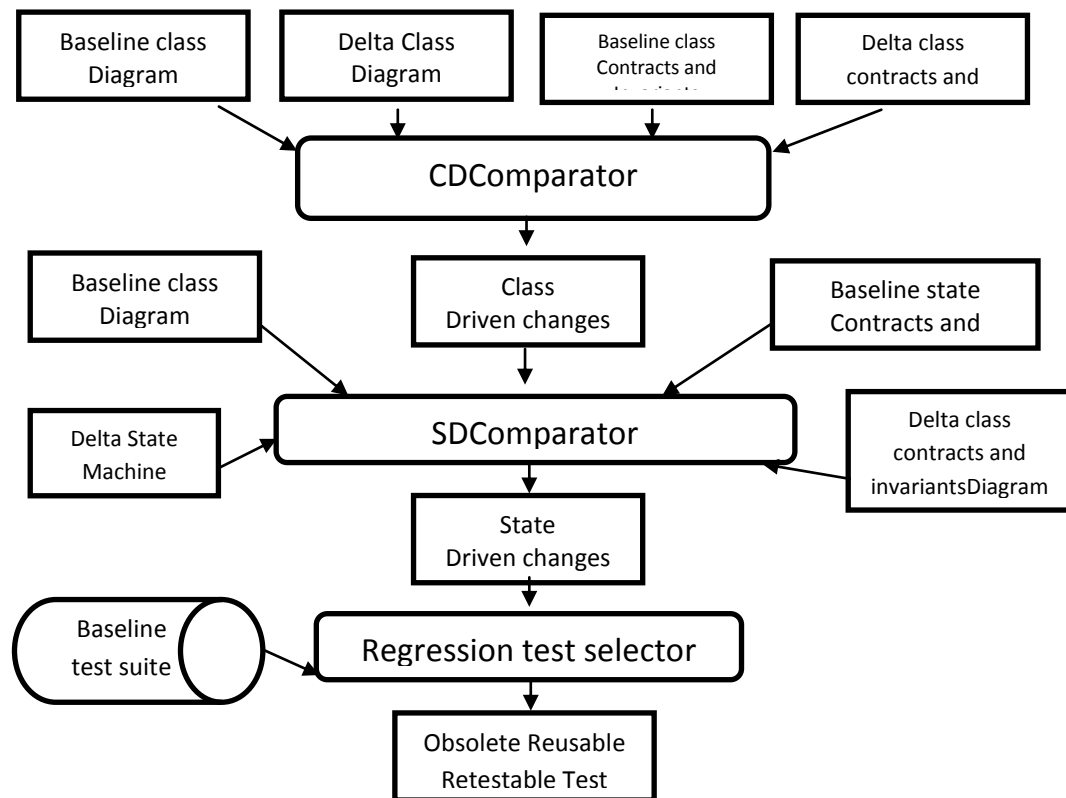
**Fig. 8: Generation of compatibility and prioritized regression test suite [25]**

Fig.8 shows the generation of compatibility and prioritized regression test suite. System integrators firstly execute the compatibility test suite related with the modified components to find out and remove the components which are not compatible. Then after selecting another component from the remaining compatible ones, runs the prioritized regression test suite to detect the integration faults. The technique was successful to solve the problem for the cases being considered and discovered few areas that are needed to be worked upon like finding out the dependency between efficiency of the prioritized test suite and its nature.

In the year 2007, Mao [27] proposed a built-in regression testing method for component based software in order to improve the ability of a component to be tested. The change information is provided to the users of the component by the developer of the component by incorporating the test script in the modified component. The test script are usually either consisting of the test cases or have the functionality to generate test cases. The component developer develops an interface for testing of the modified component and its details can be provided to the user in the form of an xml

file. The user must document the precondition of every function call can made in the proceeding test processes and after that can perform the selection of the test cases correlated with the changes in the modified component by executing the function of the interface. The technique has proved to be much more practical, improves the ability of a computer to be tested, decreased the exchange of information and reduced the cost of testing and cost of storing the information about the version of the component effectively.

In the year 2007, farooq et al. [28] presented a paper in which a model – based approach for selecting a set of appropriate test cases for regression testing was proposed which uses UML Behavioural state machines and class diagrams for recognizing the changes. The changes are categorized into two categories, class driven changes means the changes which are identified from class diagram and state driven changes means the changes which are identified from state machines. State driven changes are then used for classifying test cases into obsolete test cases, reusable test cases and retestable test cases, according to the classification scheme by Leung and White [29]. The test cases corresponding to the parts of system which are now obsolete and obsolete test cases. The test cases corresponding to the changed parts are the retestable test cases and are required to be executed again so as to ensure that the changes have been implemented properly. The test cases corresponding to parts which have not been modified are reusable test cases and are not needed to be executed again for regression testing. The fig. 9: shows the step by step process followed In the proposed approach.



**Fig: 9. State based regression testing [28]**

In the year 2007, Naslavsky and Richardson proposed an RTS approach [30] Based on control flow analysis of UML sequence diagram for model driven development environment. The technique involves a model- based transformation from sequence diagram to a CFG. The traceability between test cases and sequence diagram is used to determine which CFG elements are executed by each test case. The two CFG's originally are modified are then analysed to find out the affected model elements, and the traceability information is used to select relevant regression test cases.

In the year 2007, Ali et al.[31] proposed an RTS technique based on analysis of UML class and sequence diagram. The technique analyses classes and sequence diagram at the level of class attribute and operations. The sequence and the corresponding class diagram are analyzed and an extended concurrent control flow graph models for the original and the modified versions of the applications are then analyzed to find out the changes between program versions and to select regression test cases.

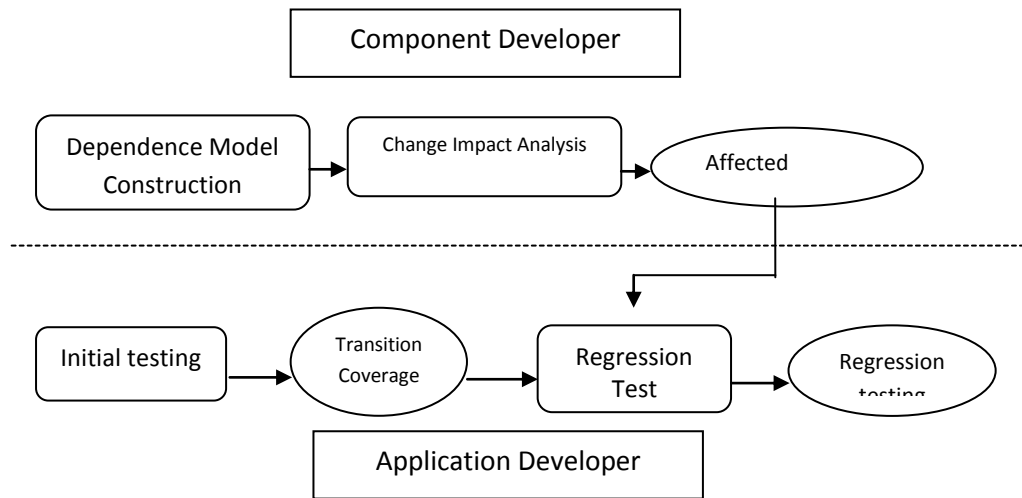
In the year 2007, Gorthi et al. [32] proposed an RTS technique based on UML use case diagrams. The approach uses the concept of behavioral slicing which decomposes use cases into user actions followed by some computations and the output and also helps in identifying changes made to the activity diagram. Whenever there is change in the requirement that will be reflected in the activity diagram also. The models for the original and the modified specifications are analysed to find the changes and select test cases for regression testing.

In the year 2011, Hagai Cibulski & Amiram Yehudai [34] proposed regression test selection technique for a test driven development. As in test driven development approach the test suit is re-executed even after a small change, and re-execute the whole test suite will lead to in increase in cost , time and effort required for regression testing. Thus an efficient regression test selection technique is required which reduces the size of the test suite in such a way that it is still able to detect a bug in the piece of code modified and provide a fault detection rate approximately equal to the rate of the executing the whole test suite. A tool named test Rank implementing test use prioritization technique which was change sensitive was developed. The technique uses dynamic program analysis and natural language analysis. The approach proved to be useful for medium scale system but is yet to be proved for large real time driven development systems.

In the year 2011, Tao, Li and Gao [35] presented a paper which focuses on the problem that whenever a Component is modified, it is not only that Component which is effected but it can be the whole system which is defected. Thus a technique which just not tests the modified Component but also the whole system was proposed. This technique involves analyzing the modification, judging the effect of that modification and updating the already existing test suite. Also a case study presented on a practical Component based system proved the technique is feasible as well as efficient.

In the year 2012, Tamil Sen and Rajib Mall [36] presented an article regarding regression test selection technique S-RTS by analyzing a Component's state and dependence model. The approach aims to reduce the size of the test suite for regression testing. Whenever a Component is modified a dependence graph is constructed for the modified Component. Then the provider of the Component figures out the information regarding the modification by analyzing the state and dependence

models of the modified Component. Then while the new version is to be released, the information regarding the affected transition [36] is provided to the user of the Component. The selections of the test cases is done by the user based on the information provided by the Component provider and the transition coverage of the original test suite which is calculated by the component user in the application environment



**Fig. 10: A Schematic of S-RTS Methodology [36]**

The technique assumes that the state model and the code of the component correspond well with each other and thus it is very easy to use the technique when the code can be produced automatically from the state model.

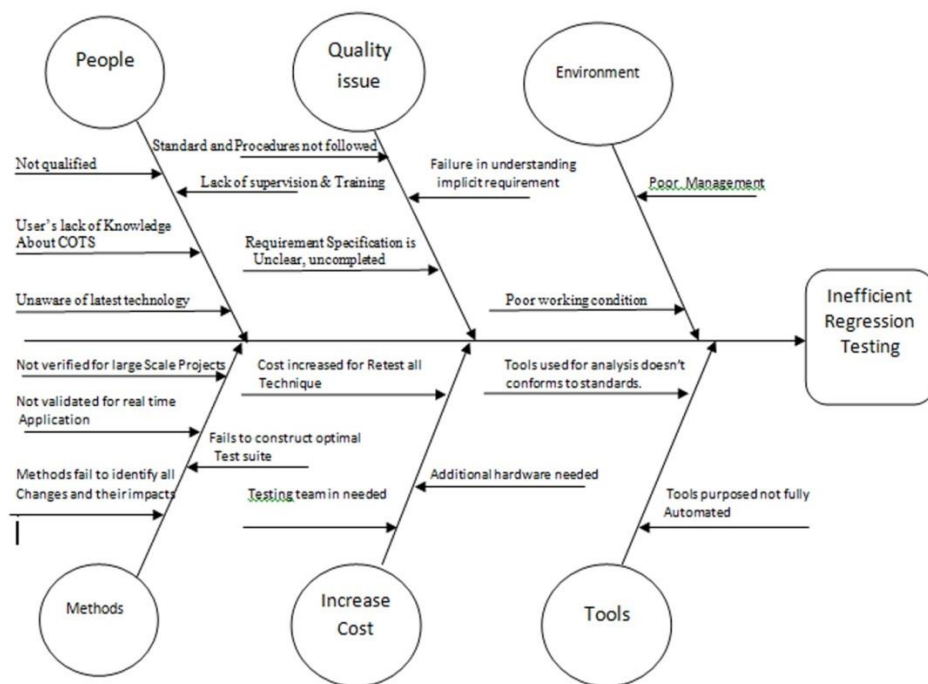
In the year 2013, ural and yenigun [37] proposed an approach for regression test suite selection using dependence analysis. Dependence analysis on an extended finite state machine representation of the requirements of the system under test is used for identifying various types of control and data dependencies between transitions caused due to set of modification on the requirements. The dependencies are used for RTS selection, by considering the coverage of dependencies related to the effects of the modifications and the dependencies identified as uncovered provide a basis for adding regression test suite with test cases covering uncovered dependencies.

**TABLE 1.SUMMARY OF SURVEY**

Reference	Analysis and comparison of Regression technique		
	Thrust area	Contribution	Results
[12]	Regression test selection For java software	First safe regression test selection technique for java software .RETEST –regression test selection system was developed	The amount of reduction in size of test suite was different for different versions
[13]	Using component meta-content to support the regression testing of component-based software	Proposed 2 approaches –code-based & specification –based test selection techniques	26% reduction in taking effort. The cost of the making meta contains available and selecting the test cases should be controlled
[19]	Regression testing for component based software systems by enhancing change information	Developers provides the enhanced change information along with the component in form of a xml file allowing user to select a subset of the existing test suite	Technique is practical & reduces testing but is to be validated for large and real time systems
[20]	Towards software architecture –based regression testing	The aim was to be provide a framework and a method for regression testing when changes have been introduced to both the software architecture and also to implementation	Does not specify how to recreate the original SA if the code no more corresponds to original SA For doing regression testing
[24]	A systematic regression testing & tool for software components	Retesting techniques dealing with API based changes & tool-COMP Test was developed	The out of date and the reusable test cases can be easily detected
[25]	Compatibility and regression testing for COTS components based software	Proposed a technique using compatibility testing and prioritized regression testing	Solved problems considered

Reference	Analysis and comparison of regression testing techniques		
	Thrust area	Contribution	Results
[27]	Built-in Regression testing for component based software systems	Proposed built in approaches, developers provides change information to the users by incorporating the test scripts in the modified components.	Enhances testability of component ,reduced the cost of testing and cost of storing the information about the version of the component
[28]	An approach for selective state machine based regression testing	UML based selective regression testing strategy using state machine and class diagram for change identification.	Validated for case study Student Enrolment system
[30]	Using traceability to support model-based regression testing	Proposed RTS approach based on control flow analysis of UML sequence diagrams	Workable and yet to be proved for real time system
[31]	Regression testing based on UML design models.	RTS techniques based on analysis of UML class and sequence diagrams	Workable but yet to be proved for large scale systems
[32]	Specification based approach to select regression test suite to validate changed software	Proposed RTS technique using UML use case &activity diagrams	Validated for medium scale systems
[34]	Regression Test Selection techniques for test driven development	Tool-Test Rank implementing test case prioritization techniques which were change sensitive was developed	Successful for medium scale systems but it yet to be proved for large real time test driven development systems
[35]	Regression Testing of Component-Based software: A Systematic Practise Based on State Testing	Proposed a technique involving analysis of the modifications, the effects of those modifications and updating the already existing test suite	Workable and efficient
[36]	State-Model-Based Regression Test reduction for component-based software	Proposed a technique S-RTS which analyse component's state and dependence model	Easy when the code can be produced automatically from the state model.

The research discover that future work is required in the fields like finding out the use of additional dynamic models for example, temporal properties [38]for constructing test suites, verifying the approaches for large scale projects and real time applications, enhancing the ability of the component to be tested, using the concept of slicing to further improve the change information, finding out ways to improve the efficiency of a test suite constructed by using test case prioritization technique and the traits of the test suite like granularity and grouping [39], to develop an automatic testing tool and to develop a common platform for component developer and user.



**Fig.11: Cause- effect diagram for inefficient regression testing**

Fig.11 shows the cause effect diagram to solve the problem of inefficient regression testing. For finding out an effective approach towards regression testing is very important first to identify the various reasons which results in inefficient regression testing. Here we are using cause –effect diagram to find out major causes which leads to an improper and ineffective regression testing of the component based software. Cause effect diagram is used so as to recognize the causes and the reasons behind those causes. The cause-effect diagram being proposed states that various causes can be people, method, tools, environment, quality issues, and increase in cost.

- i. People are a major cause, when we say people they can be testers, users of the components or the developers of the components. They are the major cause as sometimes they are not well qualified, not trained properly, not aware about the latest trends and technologies and the users of the components don't have complete knowledge about the third party components.
- ii. Another major cause is the methods or the approaches which have been proposed, as these methods have not been validated for large-scale real time applications, doesn't identify the changes and their impacts on rest of the system accurately.
- iii. Quality issues are not handled carefully such as the poor documentation, improper requirements specification, proper standards and procedures are not followed while implementing a method or an approach and the implicit requirements are not well understood. Quality issues should be looked after carefully so that the final production or the final approaches is a quality production or a quality approach.
- iv. Another factor of ineffective regression testing is use of inefficient tools or the equipment which are used for analyzing itself is not accurate the test results will defiantly be incorrect. Also various tools being proposed by the researches are not fully automated
- v. Some other causes are improper working conditions and poor management. if the working conditions are not good and proper facilities are not provided to the employees, the efficiency of working decreases.
- vi. Increased cost is one of the major causes as a testing team is required for doing effective testing, additional hardware is also needed and also retest all approach consists of re-existing test cases thus is an expensive technique and if used will lead to increase in cost of testing.

### 3.1 Research Gap Analysis

Various approaches for regression testing have been proposed based on different strategies each having merits or demerits. There are certain design model based approaches which use UML models, where each approaches uses different UML diagrams or combination of diagrams. An approach proposed in [22],[28] is used for selecting a set of appropriate test cases for regression testing was proposed which uses UML behaviour state machines and class diagrams for recognizing the changes and results in classification of test suite. The approach in [31] recognizes the change and selects the test cases by using UML class diagrams and sequence diagram. The approach in [30] uses sequence diagram for regression test selection in a model driven environment. The approaches in [32] use case diagram and activity diagrams to select the test cases. The approaches being reviewed selects or classifies the test suite but does not efficiently reduce the size of the regression test suite, also does not allow inserting new test cases into regression test suite corresponding to the new features being added in the system.

### 3.2 Problem Formulation

Component-Based Development is an approach of developing software systems by using components. The components are stored in a repository, which are useable by other programmers. Component-Based software system may contain external components as well as in-house built components. During the maintenance phase the components get altered or modified very often. When a component is altered, it is not only that component which is affected but it is the whole system which is affected. The type of testing which not only ensures that the modified component is working fine but also ensures that the changes have no adverse or severe effects on the rest of the system is called as regression testing. But due to unavailability of the knowledge about third party components it is difficult for the component users or testers to perform the testing in an efficient manner. As the component users don't have the information regarding the modifications done in the component, it creates a problem

for them to do an appropriate selection of the test cases from the original test suite for testing the altered system thus there is a need of an efficient regression test selection approach which results in a reduced and effective regression test suite which test all the changes for the system.

### **3.3 Objectives**

1. To study existing approaches being proposed for identifying and analysing and also understanding existing techniques for regression testing of component based software.
2. To propose an approach which creates a reduced and efficient regression test suite by selecting the test cases from the existing test suite and also recommends new test cases to test new added features in the system.
3. To validate the approach using the case study of automated teller machine and analyses the results.

# Proposed Methodology: Regression Test Selection and Recommendation (RTSR)

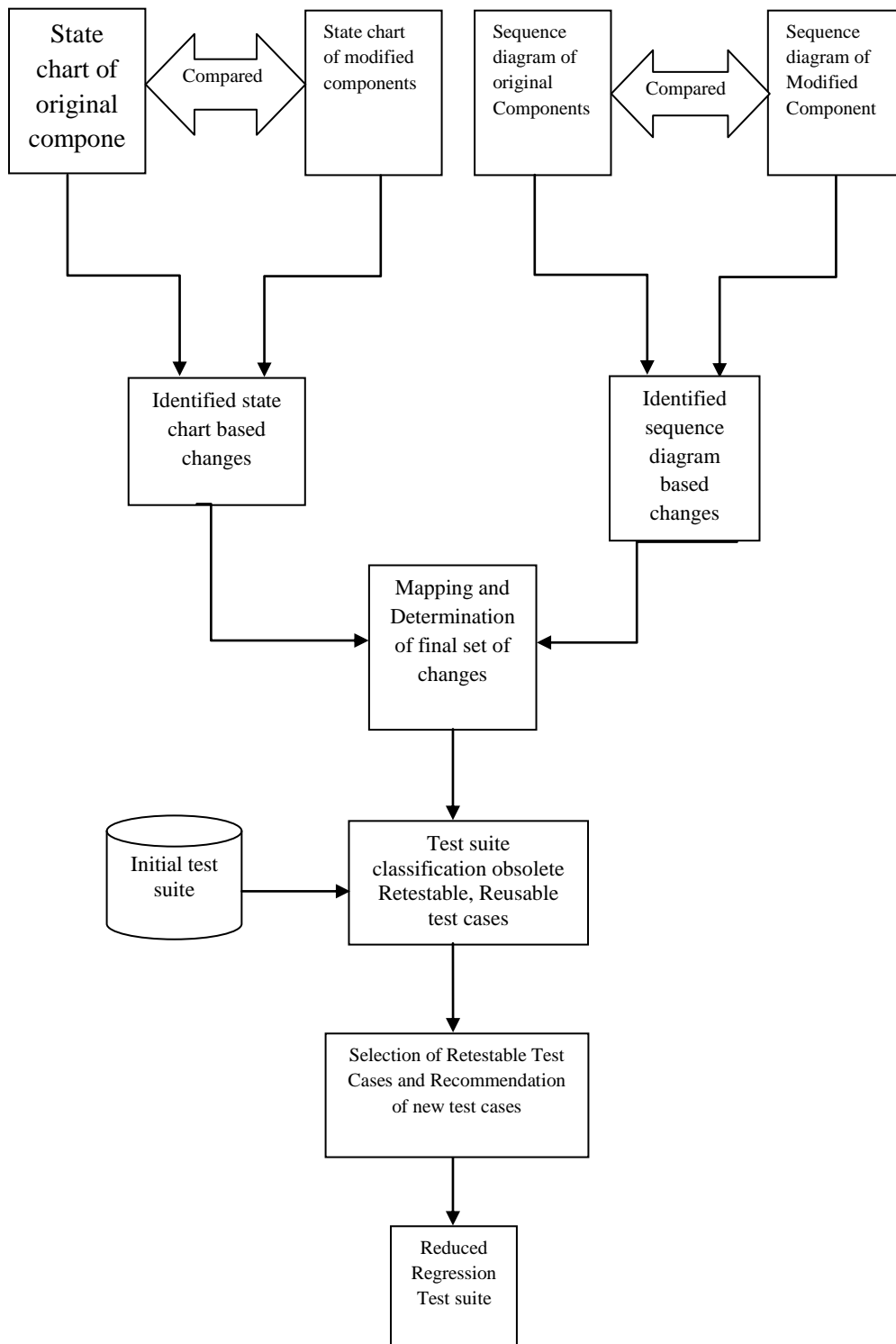
---

### 4.1 Overview

A component changes very often in maintenance phase and those changes should be tested, also it is very essential to ensure that those changes do not have any side effects rest of the system. This type of testing is called regression testing. But re-exercising all the test cases will be a very time consuming and tedious task; therefore we need an approach which can select a subset of test cases from the initial test suite which can be used to perform regression testing. As when a system is evolving it has to undergo many changes and whenever some changes and whenever some change will occur in the system the corresponding changes will be reflected at the design level also. We propose an approach “**Regression Test Selection and Recommendation**” (RTSR) Which uses UML diagrams (state chart diagrams and sequence diagrams) to investigate the changes, which are further used to select a subset of the test cases from the initial test suite and also used to recommend new test cases for regression testing of component based software. The approach RTSR identifies the changes, select test cases from the original test suite, also recommends new test cases if needed and gives regression test suite as final output.

In UML modelling different artefacts are related with each other, therefore some modification in an artefact can lead to some modification in other artefact. Therefore it is very necessary to find out the changes in those artefact's and also identify the artefact's which are indirectly affected by those changes. Thus regression test selection includes recognizing the changes in the modified system, identifying the components of the system being affected by the changes and finally selecting those test cases which correspond to changes identified.

**Fig 12. Shows the basic model of the proposed approach (RTSR), describing the flow of major steps involved in the approach.**



**Fig 12: Regression Test Selection and Recommendation (RTSR) Approach**

The basic steps used in the approach proposed RTSR are illustrated as follows:

- i.** The state chart diagram of the original component is compared to the state chart diagram of the modified system to identify the state chart based changes. Also the sequence diagram of the original system is compared to the sequence diagram of the modified system to identify the sequence diagram based changes.
- ii.** Then the sequence diagram based changes are mapped with identified state chart based change to find out the associated state chart based change for each identified sequence diagram based change and then the final set of changes is determined.
- iii.** The final analysed changes are used to classify the original test suite into three categories: obsolete test cases, retestable test cases and reusable test cases.
- iv.** Then some new test cases are recommended if required, i.e. when the original test suite does not contain a test case to test the added features of the system.
- v.** Then final step is to determine regression test suite , the retestable test cases and new recommended test cases together forms the regression test suite having less number of test cases as compared to original test suite.

#### **4.2 Detailed Description of Proposed Approach-(RTSR)**

This section illustrates the proposed approach- Regression Test Selection and Recommendation in detail by elaborating all the steps used in the approach to get the regression test suite. The section is divided broadly into two sub-sections i.e. 4.2.1 and 4.2.2. the first sub section illustrate the identification of changes in the system by comparing UML state chart and sequence diagrams of the original and modified version of the system and the second sub-section illustrate the development of regression test suite by classifying the original test suite and selecting retestable test cases and recommending new test cases.

## 4.2.1 Change identification and analysis

### Step1: Identifying and Analysing State Chart Based Changes

State chart machines describe all the states an entity undergoes and the transitions among those states which are dynamic in nature. State chart machines provide the description about the event which is caused a state change; the guard condition which must be satisfied in order to make a transition has been fired. Here we consider following changes which will be considered under the category of the state chart based changes.

- i. **Additional or Removal of a state:** if there is addition of a new state or removal of some existing state in the state chart diagram of the original component, then it will be considered as a state chart based changes and will also lead to addition or removal of certain transitions [40], hence we can consider this case as similar to addition or removal of transition and there is need to consider it as a separate change for test. Fig . 13 shows the change of addition of state leading to addition of transition.

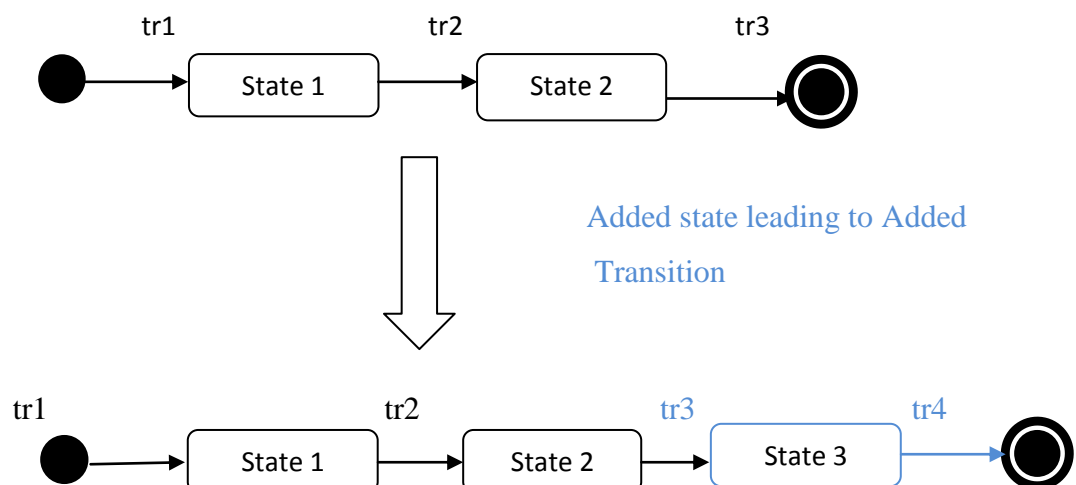
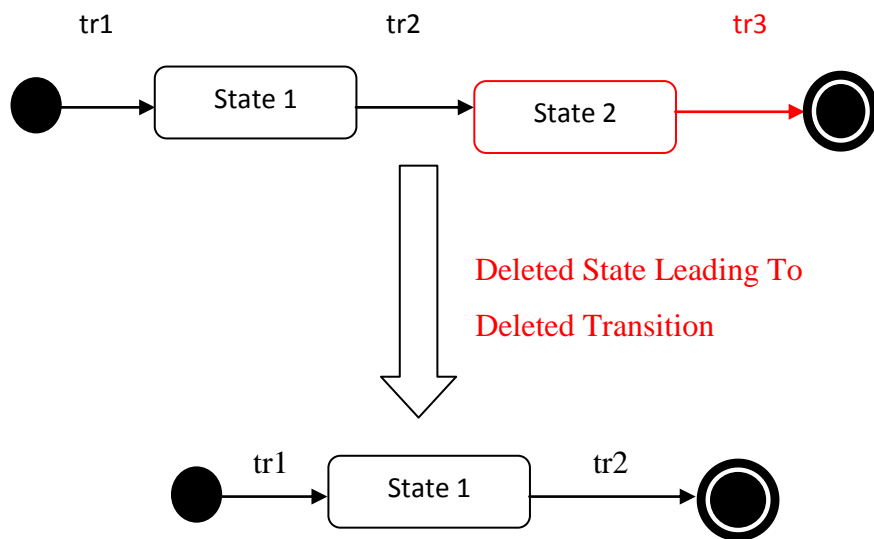


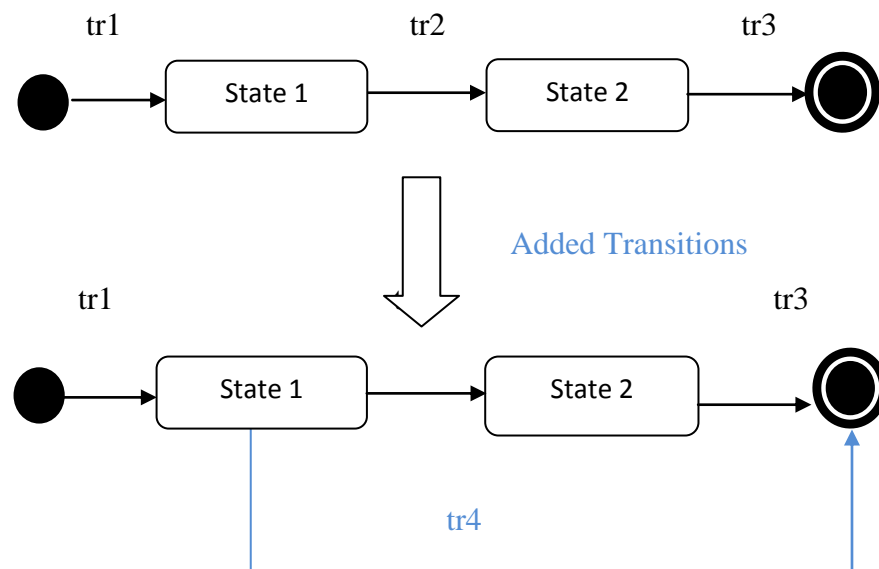
Fig .13: Addition of a state

Fig. 14 shows the change of removal of state: state2, leading to detection of a transition tr3.



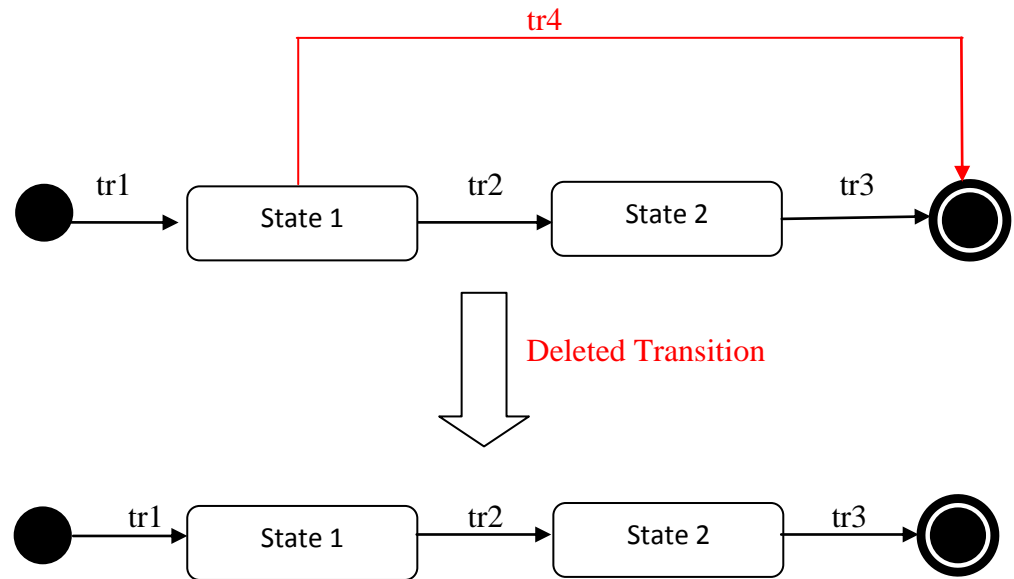
**Fig. 14: Removal of a state**

- ii. **Addition or Removal of Transition:** In case a new transition is inserted or some existing transition is deleted, we consider it as a state chart based change. The basis on which a transition is removed can be removal of any of the states between which the transition exist or can be removal of event which triggered the transition or can be removal of other artefacts of a transition such as guard condition or action [22].fig. 15 shows the change addition of a transition tr4.



**Fig. 15: Addition of a transition**

Fig.16. shows the changes of removal of a transition, t4 is deleted from original state chart diagram



**Fig . 16: Removal of a Transition**

iii. **Changed Transition:** A transition is considered to be changed if there is modification in any of the artefacts of a transition i.e. if the event which triggered the transition has changed, the guard condition which the transition must satisfy is changed, the action which will take place after the transition has occurred is changed, or there is any change in any of states between which the transition exists.

iv. **Changed Event:** the events are broadly categorized into three categories namely message events, time events and change events [22]. The reason for change in message event can be change in any of the type of these events, i.e. there may be insertion or removal of any receive event, the transition consisting such event will execute for all messages excluding the ones which are not referred by another transition having same source state. Another reason for change in message events can be change in call or signal event.

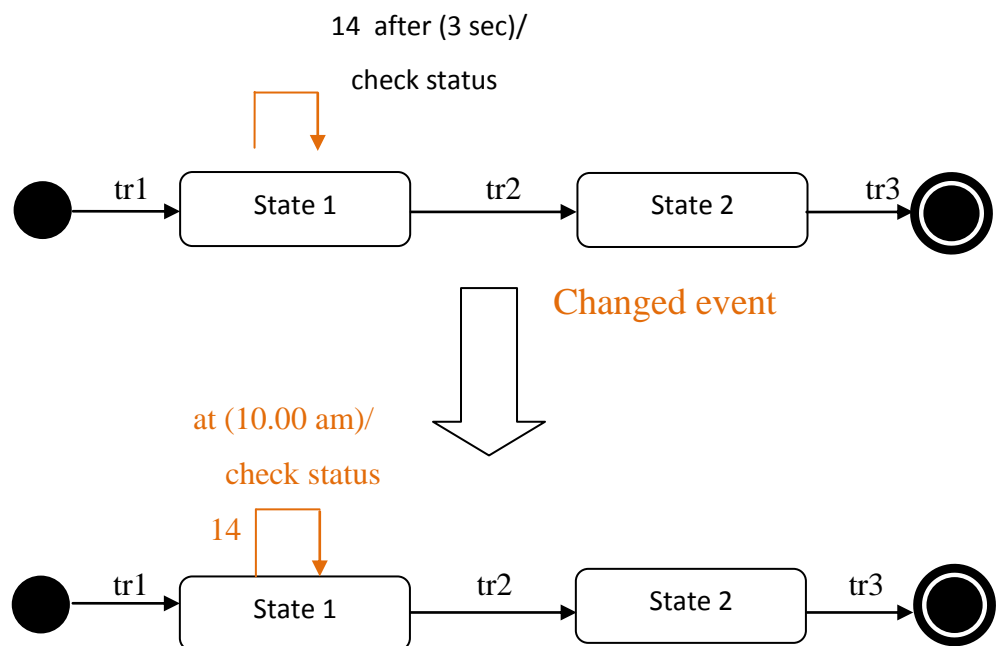
**Call events** as the name indicates are the ones which cause a method to be invoked and the reason for change in such events can be change in the

number of parameters or type of parameters the event is taking as input or the corresponding method in sequence diagram is changed, which will be explained in next section.

**Signal events** are likely call events but a major difference is that signal events are not synchronous in nature while on the other hand call events are synchronous and the reason for change can be change in number or type of parameters and are handled in same way as that of change in call events.

**Time events**, another type of event is time event- which is used for representing at what time that event must take place, which can be specified in two ways like “after” 5 seconds or “at” 10.05a.m.

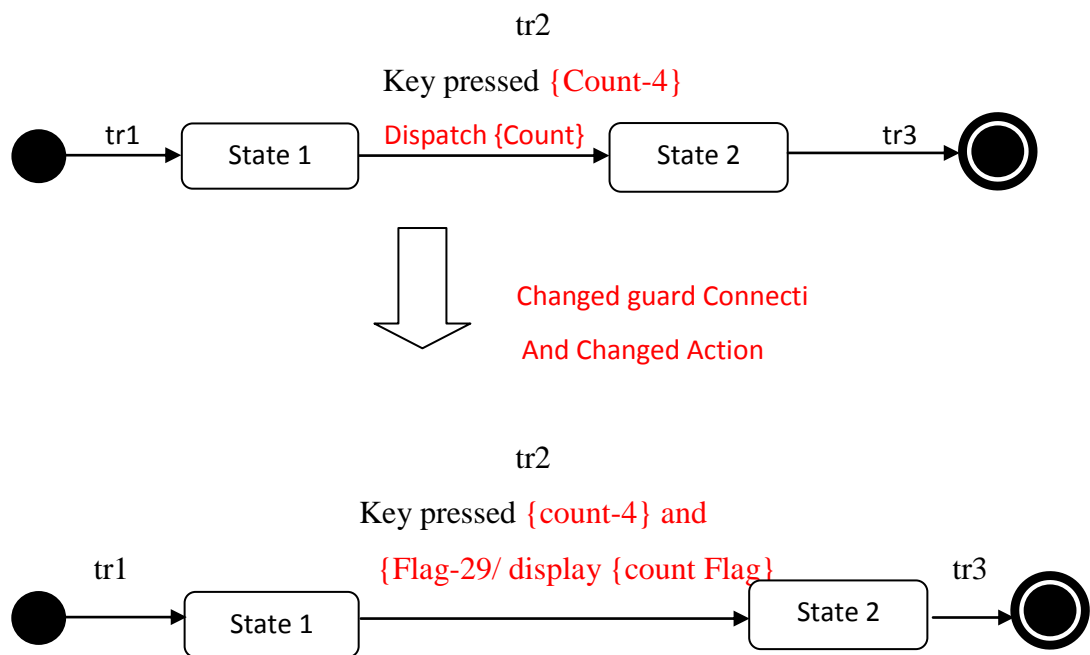
**Change event**, another type of event is change event –which are represented using “when” and then there is a condition, like when (flag=1) or when (count<4). The reason for change in time event and change event can be change in their expression having some changed or some deleted or inserted parameters. Fig. 17 shows the changes of an event of transition t4.



**Fig.17: Changed Event**

**Changed Action & Changed Guard Condition:** Guard condition is a Boolean expression which is checked, if it comes out to be true then the transition will occur followed by execution of the action, else the transition will fail. The change in action or guard can be due to change in the expression or in the parameters they are using.

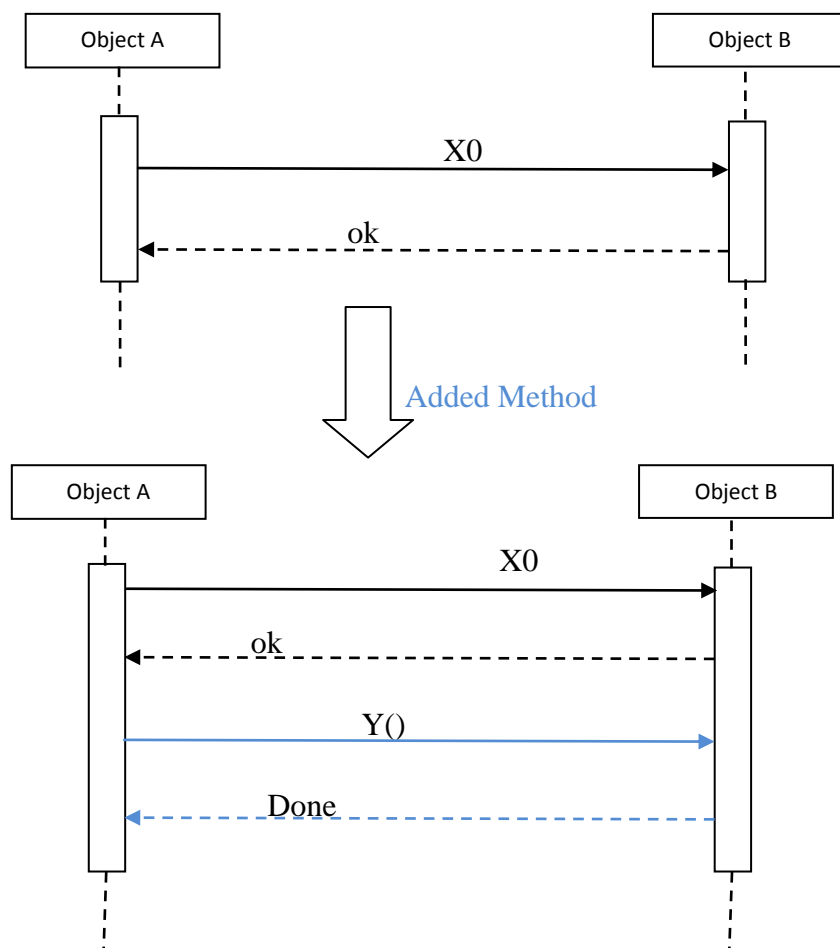
Fig.18 shows the change in action and guard condition of a transition t2, the original guard condition [count<4] and if the condition comes out to be true, the action will be display(count). Changed transition has guard condition [(count<4) and (flag<2)] and if the condition comes out to be true then action is display (count, flag).



**Fig.18: Changed Action & Change Guard Condition**

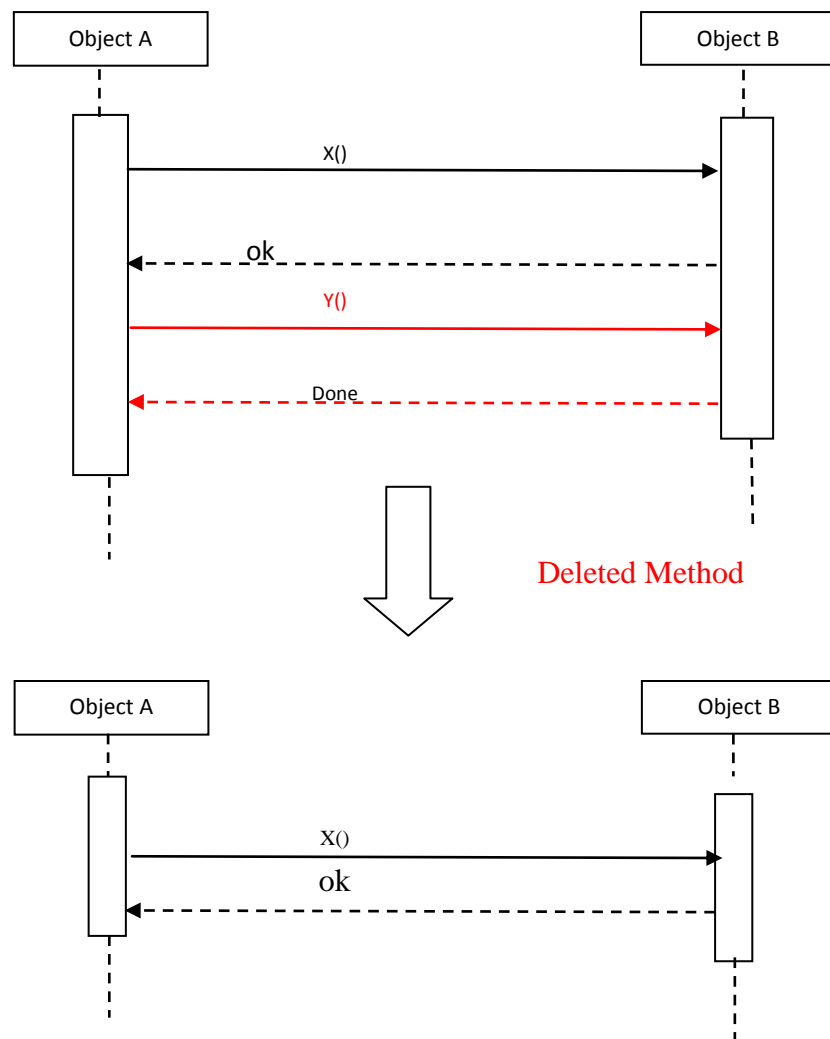
## Step 2: Identifying and Analysing Sequence Diagram Based Changes

- i. **Addition or Removal of Method:** if a method is added or removal in/from the existing sequence diagram then it is considered as a sequence diagram based change. Also when a synchronous message/method is added/ removed then a corresponding return message is also added / removed respectively therefore there is no need to consider it as a separate change to be tested. Fig. 19 shows the change of addition of the method y(), resulting in addition of return message : done.



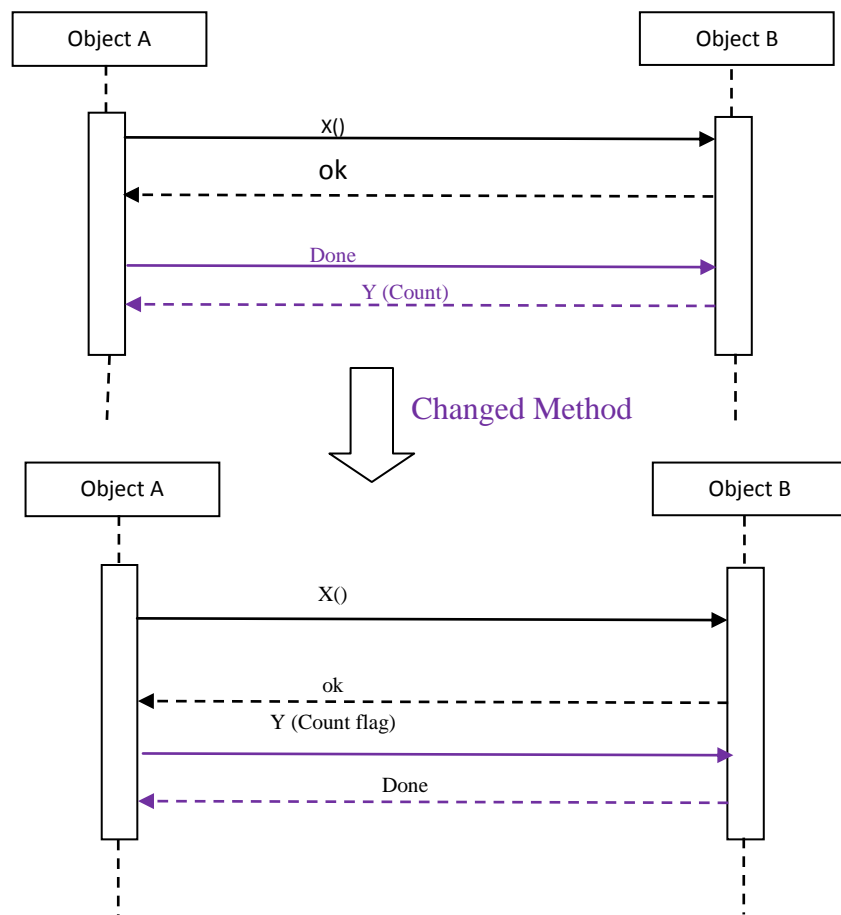
**Fig.19: Addition of Method**

Fig. 20 shows the change of deletion of a method  $y()$ , resulting in deletion of return message : done.



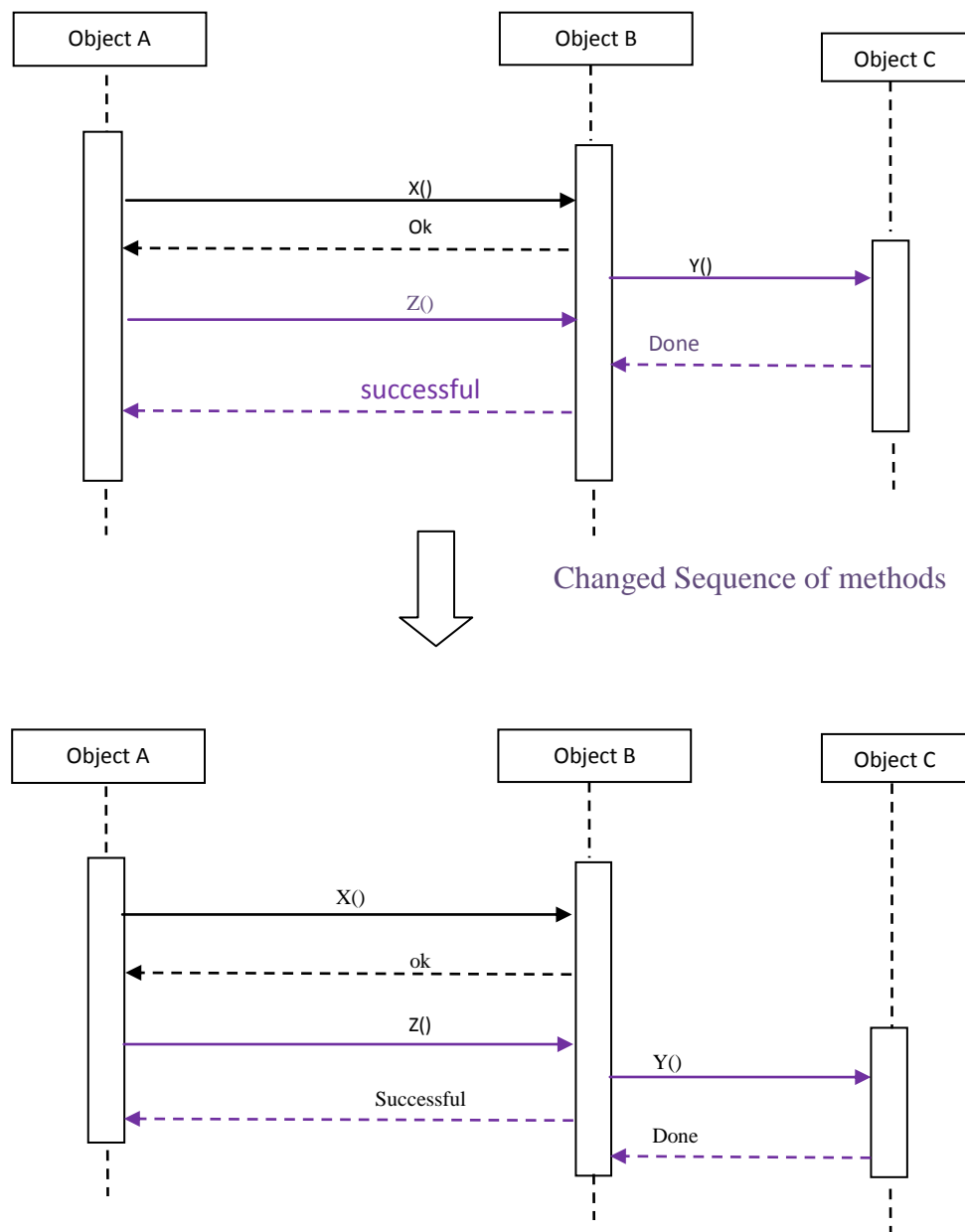
**Fig.20: Removal of Method**

- ii. **Modified Method:** A method when modified is considered as a sequence diagram based change, the reason for change in the method can be change in number of parameters it takes as arguments or change in the objects between which a method exist or change in the sequence of the messages or there can be some guard condition added or removal or changed in the method. Fig.21 shows the modified method y() , i.e. from y(count) to y(count, flag).



**Fig. 21: Modified Method**

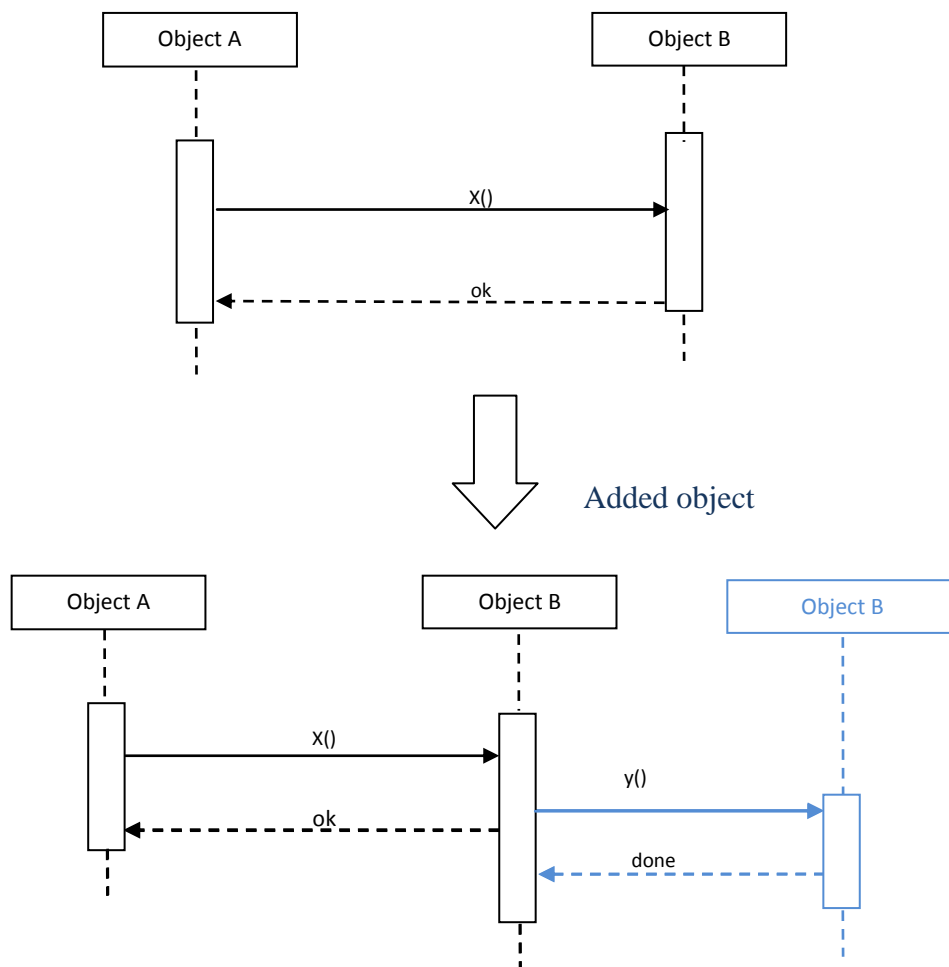
Fig.22 shows the change in sequence of methods earlier sequence is x(), ok, y(),done ,z(), successful and later it is x(), ok, z(), successful , y(), done .



**Fig.22: Modified Sequence of Methods**

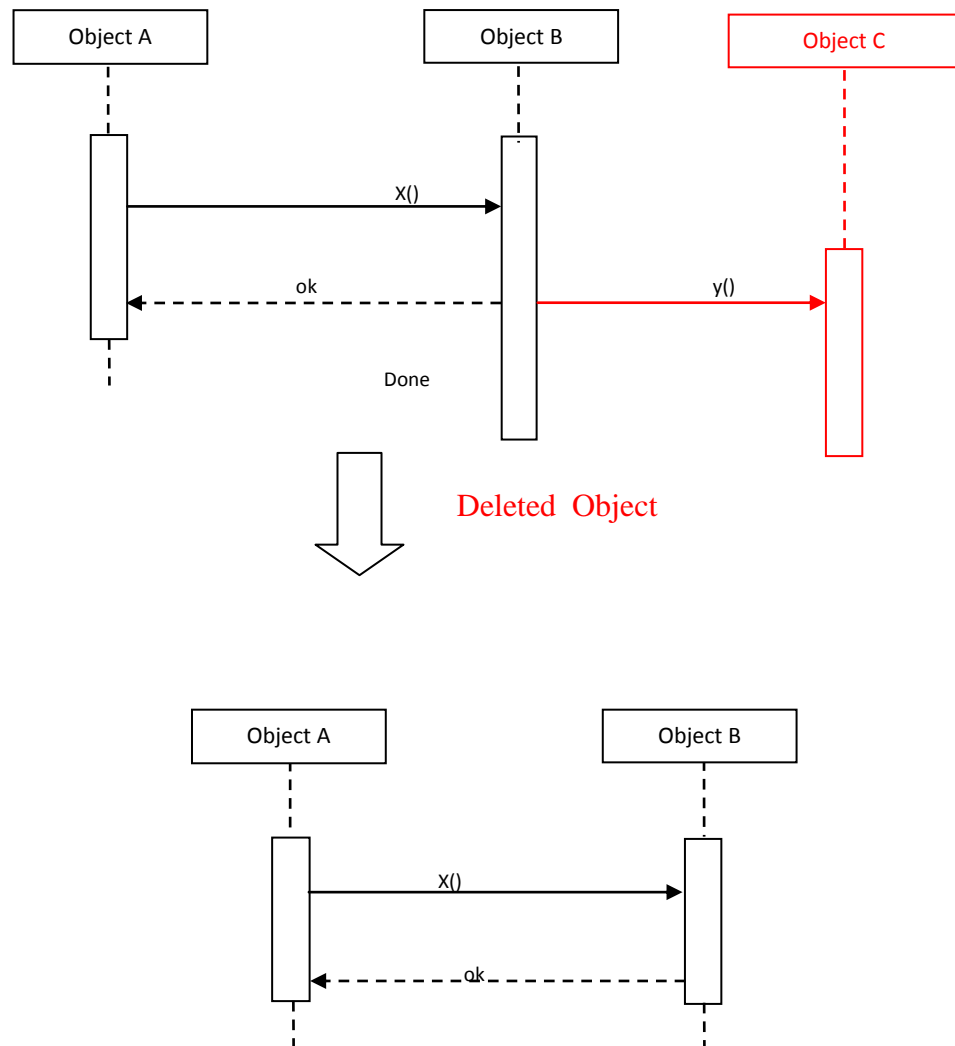
- iii. **Addition or Removal of Object:** if a new object is added or some existing object is removed from the sequence diagram due to may be addition or removal of some component in the system then such changes are considered under the category of sequence diagram based changes but as addition/removal of objects leads to addition/removal of some method therefore there is no need to consider it as a separate change for testing.

Fig. 23 shows the change of addition of an object of type C resulting in addition of method y (), leading to addition of return message done.



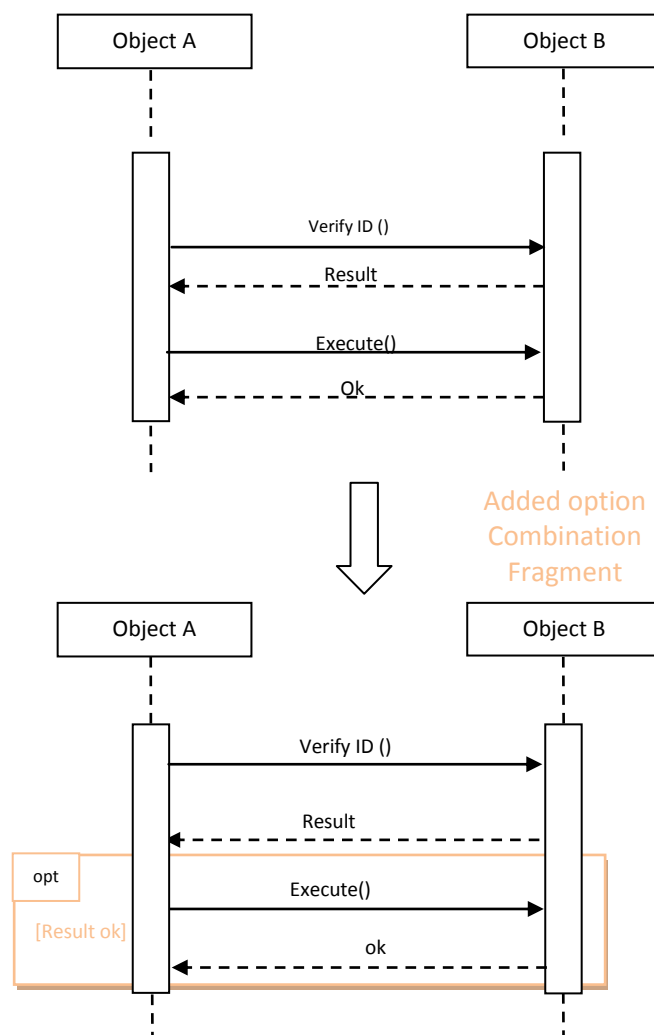
**Fig .23: Addition of Object**

Fig. 24 shows the change of deletion of an object of type C resulting in deletion of method y() , and return message done.



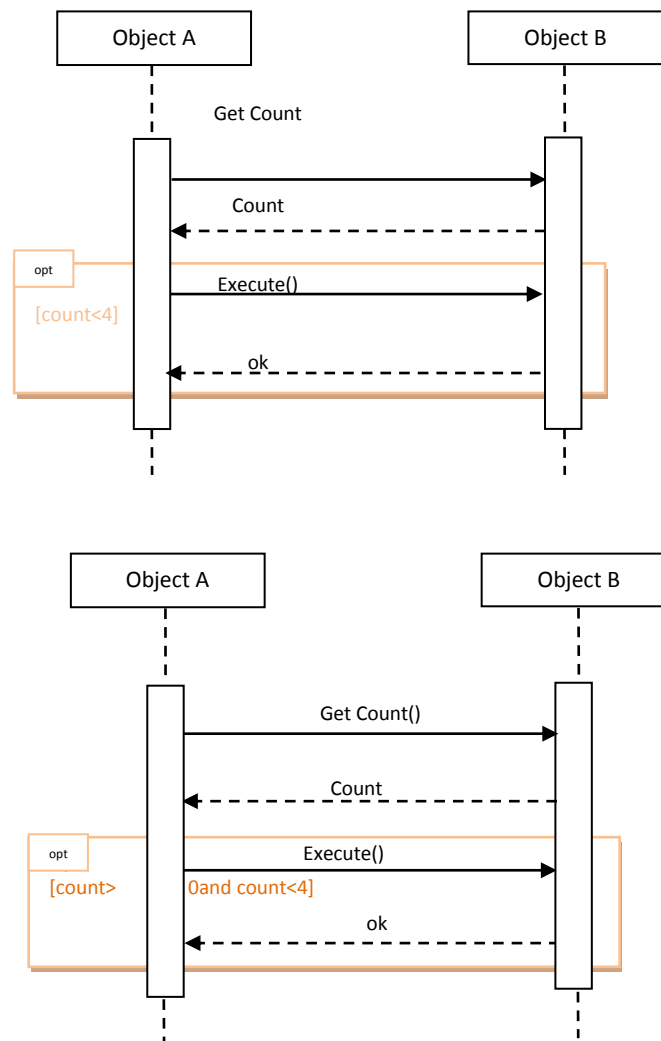
**Fig.24: Removal of Object**

- iv. **Addition or Removal of Combination Fragments:** In sequence diagram there are three types of combination fragments [41] i.e. loops, opt means options and alt means alternatives. Alternatives allow the modelling of “if then else” logic, an option is used to model “if then” statement and loops are used to model repetitive sequence and the loop’s guard is placed towards the top left corner of the fragment. Addition or removal of any of the combination fragment is considered as a sequence diagram based change. Fig.25 shows the option combination is added, with a guard condition [result = ok] i.e. if the result is ok then only the methods in the fragment will be executed.



**Fig. 25: Addition of Option Combination Fragment**

- v. **Changed Combination Fragments:** A change in a combination fragment is considered as a sequence diagram based change, the reason for change in a combination fragment can be either change in the guard condition if any or change in the methods in the fragments. Fig. 26 shows the change in guard condition of option combination fragment, earlier the guard condition is [count<4] and later is has changed to [count>0 and count<4], i.e. now the methods inside the fragment will be executed only if the value of count is gather than 0 and less than 4.

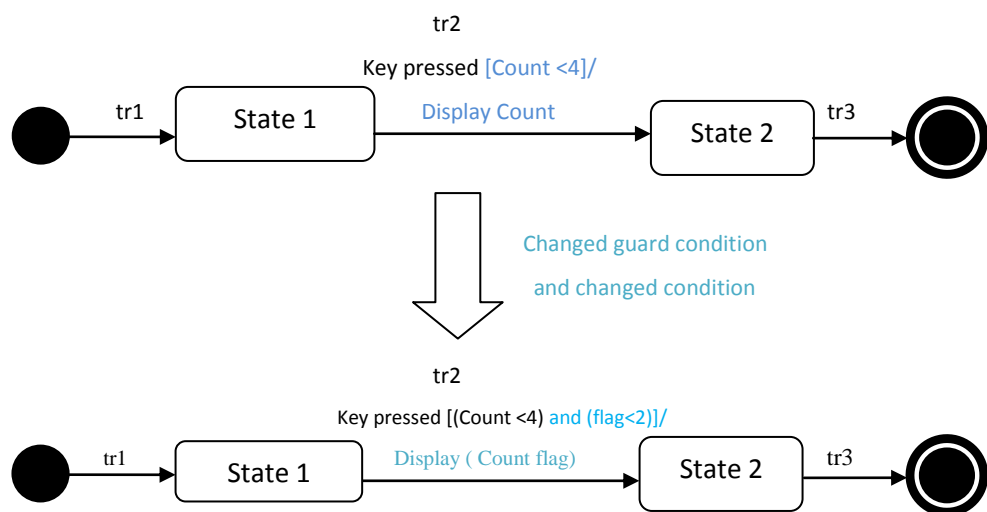


**Fig. 26: Changed Option Combination Fragment**

### Step 3: Mapping the Sequence Diagram Based Changes with State Chart Based Changes.

After the changes in the state chart diagrams of the system and the changes in the sequence diagrams of the system have been identified and analysed, the two analysed sets of changes are mapped with each other to determine the associated state chart based change corresponding to the sequence diagram based changes.

Consider the example of the system: system A to understand how the approach works, the fig. 27. Which shows state chart diagram of original and modified systems for system A, showing that there is change in guard condition leading to change in transition tr2.



**Fig. 27: change in the state chart diagram for system A.**

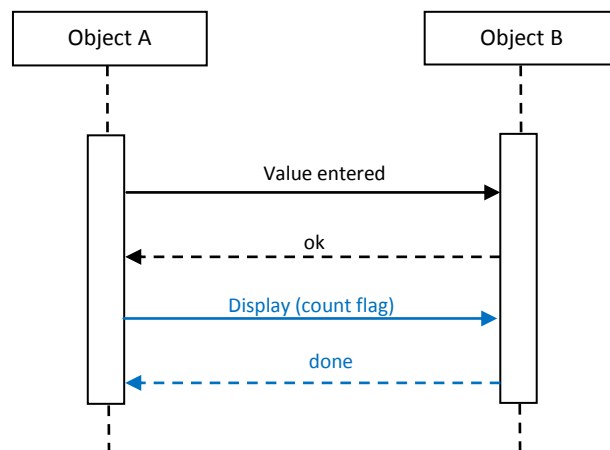
#### Identified State Chart Diagram Based Change (SC.1)

- Changed Transition tr2 – Value Entered/ Display (count, flag)

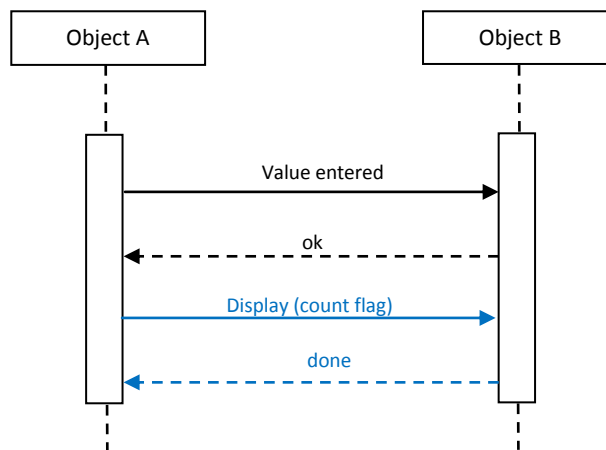
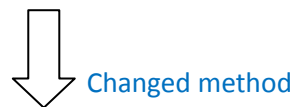
Changed Transition	tr2 value entered /display(count ,flag)
--------------------	---

The fig. 28 on next page shows sequence diagram of the original and modified system for a scenario of system a, showing that there is change in display method earlier which was taking count as parameter, later it is having two parameters namely count and flag.

The method display has changed, from display (count) to display (count, flag) as shows below.



Sequence Diagram of original system



Sequence Diagram of Modified System

**Fig. 28: Change in the Sequence Diagram for System A**

### Identified Sequence Diagram Based Change:

- Changed Method-Display (count, flag)

Changed Method	Display (count, flag)
----------------	-----------------------

### Mapped Sequence Diagram Based Change with State Chart Based Change:

Now mapping the two identified changes, the sequence based change can be described in the terms of transition of state chart diagram as:

**Sequence Diagram Based Change:** Changed Method – Display (count, flag)

**Associated State Chart Based Changes (SC.2):** Changed Transition tr2 – value

Entered /Display (count, flag)

### Step 4: Determine Final Set of Changes

Now when the sequence diagram based changes are mapped with the state chart based changes to get a set of associated state chart based changes (SC.2) we will take union of (SC.2) and analysed state chart based changes (SC.1) to determine final set of changes. Here we have got the following results after taking union as the final changes.

**TABLE II: Final Change Identified for System A**

Deleted Transition	None
Added Transition	None
Changed Transition	Test2

### 5.2.2 Development of Regression Test Suite

The initial test suite being developed for the original system is used and from that the proposed approach selects a subset of test cases to test the change. Test cases exist in from of a table having fields namely the test case id, purpose of the test case i.e. which functionality of the system will the particular test case will test and the

transitions with which the test case is associated. The Table II shows the structure of the test suite table having two randomly created test cases:

**TABLE III: Test Suite\* for System A**

Test Id's	Purpose	Transition Supplementary
Test1	For test show	tr2
Test2	For test selected purpose	tr1, tr3

\*The values entered in this table are randomly entered to just explain the approach

### Step1. Classification of Initial Test Suite

The initial test suite is classified into three categories three categories i.e. **Retestable Test Cases**, **Obsolete Test Cases** and **Reusable Test Cases** [29] on the basis of final set of changes being determined:

- i. **Obsolete Test Cases:** the test cases which are not required to be executed to test the changed system are obsolete test cases as these test cases are related with the obsolete component of the system i.e. the parts which no more exist in the new system. The test cases which contain at least one of the deleted transitions in their associated transition will be selected and are categorized as obsolete test cases.
- ii. **Retestable Test Cases:** The test cases which are not required to be executed again are retestable test cases, because these are related with changed components of the system. The test cases which contain at least one of the changed transitions in their associated transition will be selected and are categorized as retestable test cases.
- iii. **Reusable Test Cases:** The test cases which can be reused are reusable test cases as these test cases are related with the components of the system which have not been changed or modified. All the test cases in the initial test suite except obsolete test cases and retestable test cases are reusable test cases. As these test cases do not correspond to any change in the system, they are not required to be executed again but are still valid for new system.

As the change identified listed in table II, is of changed transitions only, hence the retestable test cases hence after classifying Test suite shown in Table III we get the results as retestable test case is Test1 as test case Test1 contains changed transition tr2 and rest i.e. Test2 is Reusable Test Case, shown below:

**Retestable Test Case:**

*TestCaseId:*Test1*Purpose:* To test display

**Reusable test Case:**

*TestCaseId:* Test2 *Purpose:* To test some function

**Step 2: Improvement of New Test Cases**

As there may be a possibility that some functionality is added in the system and the original test suite does not contain any test case to test that added functionality or added component, then the need arises to create new test cases, which will be included in regression test suite.

Here in the system A being considered, there are no added transitions in the final changes being identified therefore, there is no need to add new test case.

**Step3: Purpose of Regression Test Suite**

The test suite for regression testing must contain only those test cases which test the change in the system so as to perform an efficient regression testing i.e. use less time and resources and at the time also reveals all the faults.

**As Obsolete Test Cases** are the test cases which are related with those parts of the system that no more exists, therefore they are no more required and not valid anymore, hence they are **not included** in regression test suite.

**As Reusable Test Cases** are the ones which are related with the components of the system which have not been modified, therefore they are not required to be executed

again for regression testing as they do not relate to any change and hence they are **not included** in regression test suite.

As **Retestable Test Cases** are related with changed components of the system therefore these are required to be executed for regression testing and hence they are **included** in regression test suite.

**Recommended Test Cases** are the new added test cases which are required to test the new features are **included** in the regression test suite.

Thus the regression test suite will contain union of retestable test cases and added test cases, but as the for system A being considered in previous steps, there is no added test case, hence the regression test suite will contain retestable test case as shown below:

**TABLE IV: Regression Test Suite for System A**

<b>TestCaseId</b>	<b>Purpose</b>
Test1	For test display

## Chapter 5

# Implementation and Experimental Results

---

### 4.1 Overview

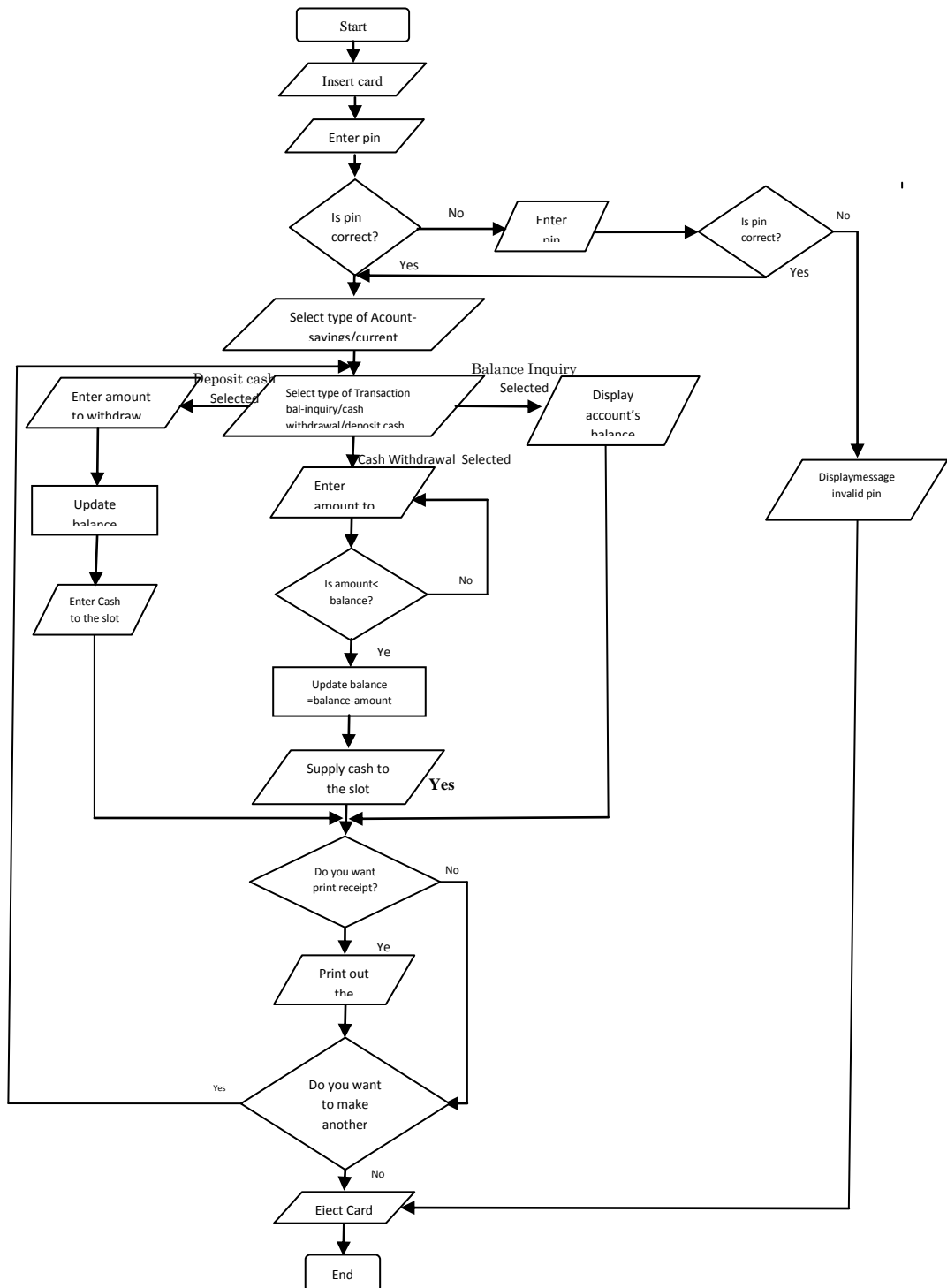
The case study of Automated Teller Machine (ATM) is being used to validate the proposed approach (RTSR). Before going into the details of the approach, let us have an overview of the system being considered so that we know the details about the functions it provides to the users, the components of the system, also some details regarding the working of system which are in our context. The user or customer is able to perform functions such as withdraw cash, deposit cash, make a balance inquiry, get a printed receipts of the transaction.

The account can be of either savings type or of current type. User or customer is authenticated before it is allowed to make a transaction, by verifying the pin and is allowed to make only two attempts with wrong pin, if the user enters a wrong pin consecutively third time then the card is ejected out of the ATM and an error message of wrong pin or invalid pin is displayed. The system includes a user interface for interacting with the user, a slot for accepting and ejecting out the card, a slot for depositing cash and a slot for ejecting out the printed receipt of the transaction. Some key terms related to the system are discussed as follows:

- i. **Term:** when the user inserts the card a term starts and ends when the card is taken out of the ATM. All the actions a user performs are done in a term, which may involve user authentication or performing any kind of a transaction i.e. withdrawing of cash, making a balance inquiry and depositing some cash.
- ii. **Transaction:** Any action which involves bank for its completion is known as a transaction of ATM. The system considered allows a user to make three kinds of transactions i.e. cash withdraw, balance inquiry, deposit cash.

- iii. **User Authentication:** user authentication involves verifying the pin which the user has entered, if the pin is verified then user can continue further to make a transactions but in case the user enters a wrong pin a second chance is given to the user to enter the pin again which is then verified but if for the third time consecutively a wrong pin is entered then the user is not allowed to continue further, the card is ejected out and an error message of invalid pin is displayed on the screen.
- iv. **Balance Inquiry:** if the user selects balance inquiry option, the screen displays the balance in the account.
- v. **Cash Withdrawal:** if user selects cash withdrawal option, the screen displays a message asking for the amount which the user wishes to withdraw. User has to enter the amount then a check is performed that the account contains balance greater than the amount being requested by the user. If the account contains balance less than the amount to be withdrawn than an error message of insufficient available balance is displayed on the screen and the user is asked to enter a valid amount again. If the accounts contain balance greater than the amount to be withdrawn then the account's balance is updated i.e. the withdrawn amount is deducted from the balance and the cash is dispensed out followed by a message on the screen requesting the user to take out the cash.
- vi. **Deposit Cash:** if user selects deposit cash option, the screen displays a message asking for the amount which the user wishes to deposit. User enters the amount and a message requesting user to deposit the cash into the slot is displayed. The user enters the cash into the deposit slot and the account's balance is updated i.e. the deposited amount is added to the balance.

**Flow chart of ATM:** The Fig.29 flowchart shows the ATM describing the sequence of steps or the whole sequential process of ATM system:



**Fig. 29: Flowchart of ATM System**

The step by step description of flowchart of ATM system is illustrated as follows:

- i. The first step which the user has to perform is to insert the card into the slot where the card is to be inserted.
- ii. Then a message is displayed on the screen requesting the user to enter pin followed by which the user enters the pin. The pin entered is then verified, if the pin entered is correct then user can continue further otherwise if wrong pin is entered then user is asked to enter pin again which is then checked but if the users enter wrong pin for the third time then an error message of invalid pin is displayed and then the card is ejected.
- iii. Once the pin is validated, then the user is asked to select the type of account i.e. saving account or current account.
- iv. When the type of account is selected, then a selection of type of transaction is to be done i.e. cash withdrawal or deposit cash or balance enquiry.
- v. If user selects balance enquiry option, then screen displays the balance in the account.
- vi. If user selects cash withdrawal option, then the user is asked for the amount to be withdrawn followed by which users enter the amount then a condition is checked that the account's balance is greater than the amount being requested by the user. If the accounts contain balance less than the amount to be withdrawn then an error message of insufficient available balance is displayed on the screen and the user is asked to enter the amount again. If the account contains balance greater than the amount to be withdrawn then the account's balance is updated i.e. the withdrawn amount is deducted from the balance and the cash is supplied to the slot which dispenses out cash.
- vii. If user selects deposit cash option, the screen displays a message asking for the amount which the user wishes to deposit. User enters the amount, the account's balance is updated i.e. the deposited amount is added to the balance and user is required to deposit the cash into the appropriate slot.

- viii. After the transaction is done, user is asked about whether he needs a printed receipt of the transaction being done. If the user selects yes than the receipts is printed, if no then control moves to next step.
- ix. Then the user is asked about whether he wants to do another transaction, if yes then control goes to the step IV of asking about the type of transaction and all the steps are to be performed in the same sequences as described above, but if the user enters no then the card is ejected.

### **Change in the Original System:**

When a system is in its maintenance phase it has to go through a lot of changes according to the changing needs and requirements of the user. The changes required to meet the desired functioning of the system as explained before are described as follows:

- i. **User authentication:** the user authentication requires that if a user enter the wrong pin instead of ejecting out the card at the first time, a check must be kept at the number of attempts a user has made so that if a user has made three wrong attempts then only the card must be ejected out and a message of invalid pin must be shown on the screen. While if the count of number of attempts has not reached to three the system must again ask the user to enter the pin and must verify it again so that user can continue to use the system.
- ii. **Print Receipt:** the another change in the previous version of the system required is a new functionality of giving the user an option to get a printed receipt of the transaction made, so that details of the transaction can be known to the user. If the user selects yes, then the receipt is printed and ejected and the user is asked about whether he wishes to continue to make another transaction or not and finally the card is ejected. Also if a user wants to simply just perform the transaction and does not want a print receipt that option must also be available i.e. if the user selects no, then the user is asked about whether you want to continue to make another transaction or not and the card is ejected.

- iii. **Cash withdrawal:** the transaction of cash withdrawal need to be changed as earlier it was not making changes in the balance of the account (saving account) leading to an incomplete and incorrect transaction.  
4EThus after a transaction of withdrawn must be detected from the account's balance and written to the database of bank.
- iv. **One Account of One User in the Bank:** Another change was that a user can have only one account in the bank, earlier the system provides the functionality to allow another transaction with another account therefore when user selects yes to continue another transaction systems asks users to select type of account in the bank when the user selects yes to continue another transaction then system asks user about type of transaction user want to do and not about the type of account.

## **5.2 Application of Proposed Approach: Regression Test Selection and Recommendation (RTSR)**

In order to validate the approach, let us consider the ATM system being illustrated in the previous section. The following section explains how the proposed approach (RTSR) is used to test the change being mentioned in the previous section. In order to use the approach we have to perform certain steps which are as follows:

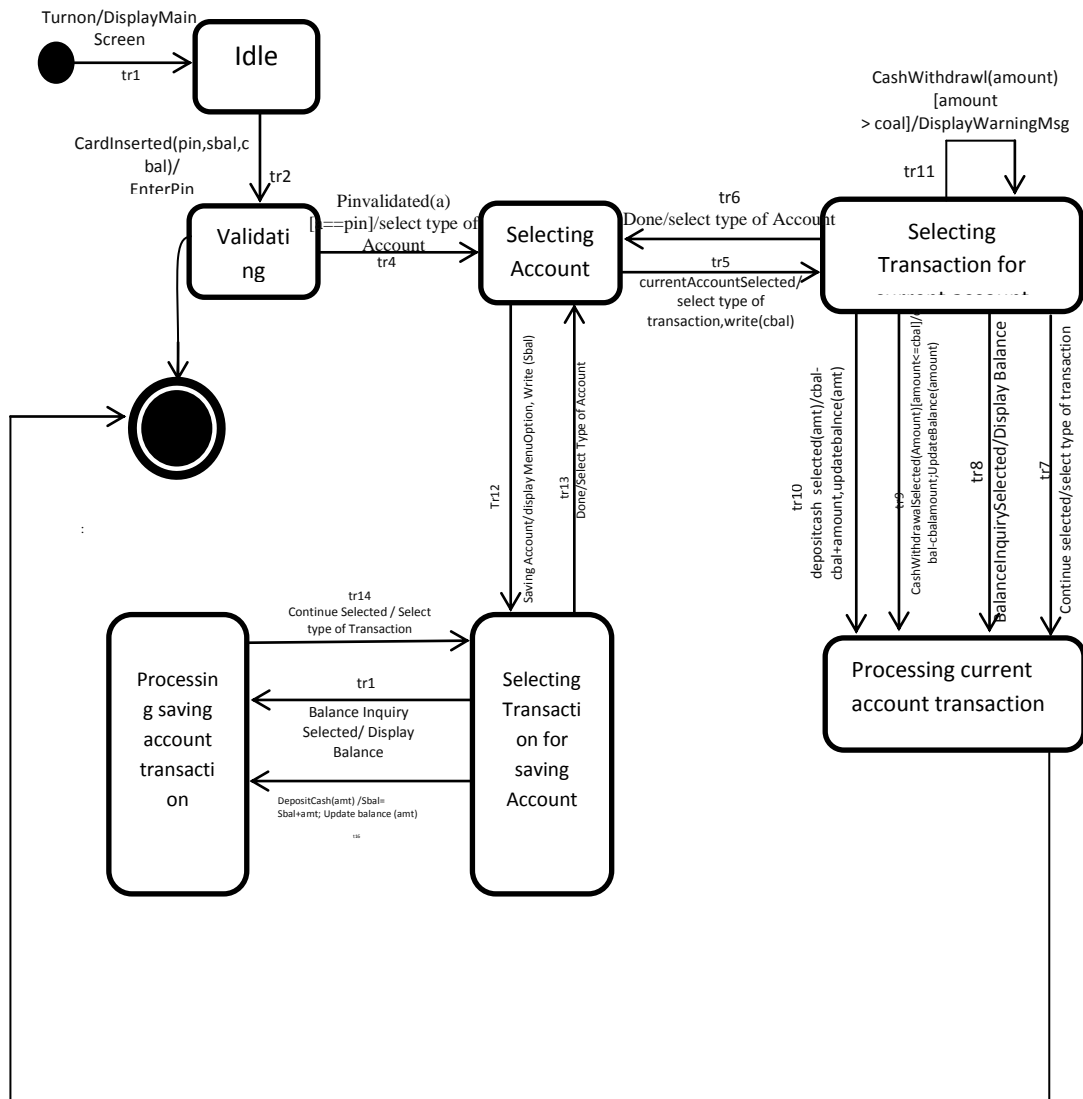
### **Step1: Change Identification and Analysis**

The first step is to identify and analyse change, which will be used further to develop regression test suite in order to do so the steps which are required to be performed are illustrated below:

#### **Step1.1: To Identify and Analyse State Chart Diagram Based Changes**

The first step is to identify the changes in state chart diagrams of the original and modified system by comparing the two state chart diagrams and then analysing the changes to determine set of state chart based changes of the system being considered.

Fig. 30. Shows the state chart of the original ATM describing the various State of the machine, the transactions between those states and the events which triggers the transactions, the guard conditions which if evaluates to be true then transaction will get fired and the resulting action which will take place after a transaction is fired.



**Fig. 30: state chart diagram of original ATM**

The details about how the machine changes from one to another state are illustrated as follows:

- i. **Idle:** the machine when turned on is in idle state and displays main screen and waits for user to perform an action.
- ii. **Validating:** when the card is inserted the machine the state is changed to validating, asking the user to enter pin.
- iii. **Selecting Account:** After that if the pin is validated the state changes to selecting account, asking user to select type of account – savings or current. Also if user has done a transaction then also machine switches to this state to allow a user to make another transaction with different account.
- iv. **Selecting Transaction for Current Account:** when user selects current account, state changes to selecting transaction for current account and asks user to select type of transaction-cash withdrawal, cash deposit or balance inquiry and also allows user to continue another transaction with same account. Also if a user selects cash withdrawal a check is performed that balance in the account is greater than amount to be withdrawn. If not machine continues to be in the same state.
- v. **Processing Current Account Transaction:** after the user has selected type of transaction, machine switches to his state and does the required processing for the type of transaction selected.
- vi. **Selecting Transaction for Saving Account:** when user select saving account, state changes to selecting transaction for saving account and asks user to select type of transaction- cash withdrawal, cash deposit or balance inquiry and also allow user to continue another transaction with same account. If a user selects cash withdrawal a check is performed that balance in the account is greater than amount to be withdrawn, if not machine continues to be in the same state.
- vii. **Processing Saving Account Transaction:** After the user has selected type of transaction, machine switches to this state and does the required processing for the type of transaction selected. Here if the cash withdrawal is selected processing just involves checking the sufficiency of balance and not updating the balance in the account.

viii. **Final State:** The machines comes to final state either when transaction processing is finished and user finished and user wishes to exit resulting in ejecting card and displaying main screen or when wrong pin is entered followed by actions ejecting card , display invalid pin message and display main screen.

Fig. 31: Shows the state chart diagram of the modified ATM describing the various states of the machine and the transactions between those states. To identify the changes, the two state chart diagrams Fig. 30 and Fig. 31 of original and modified system respectively are compared.

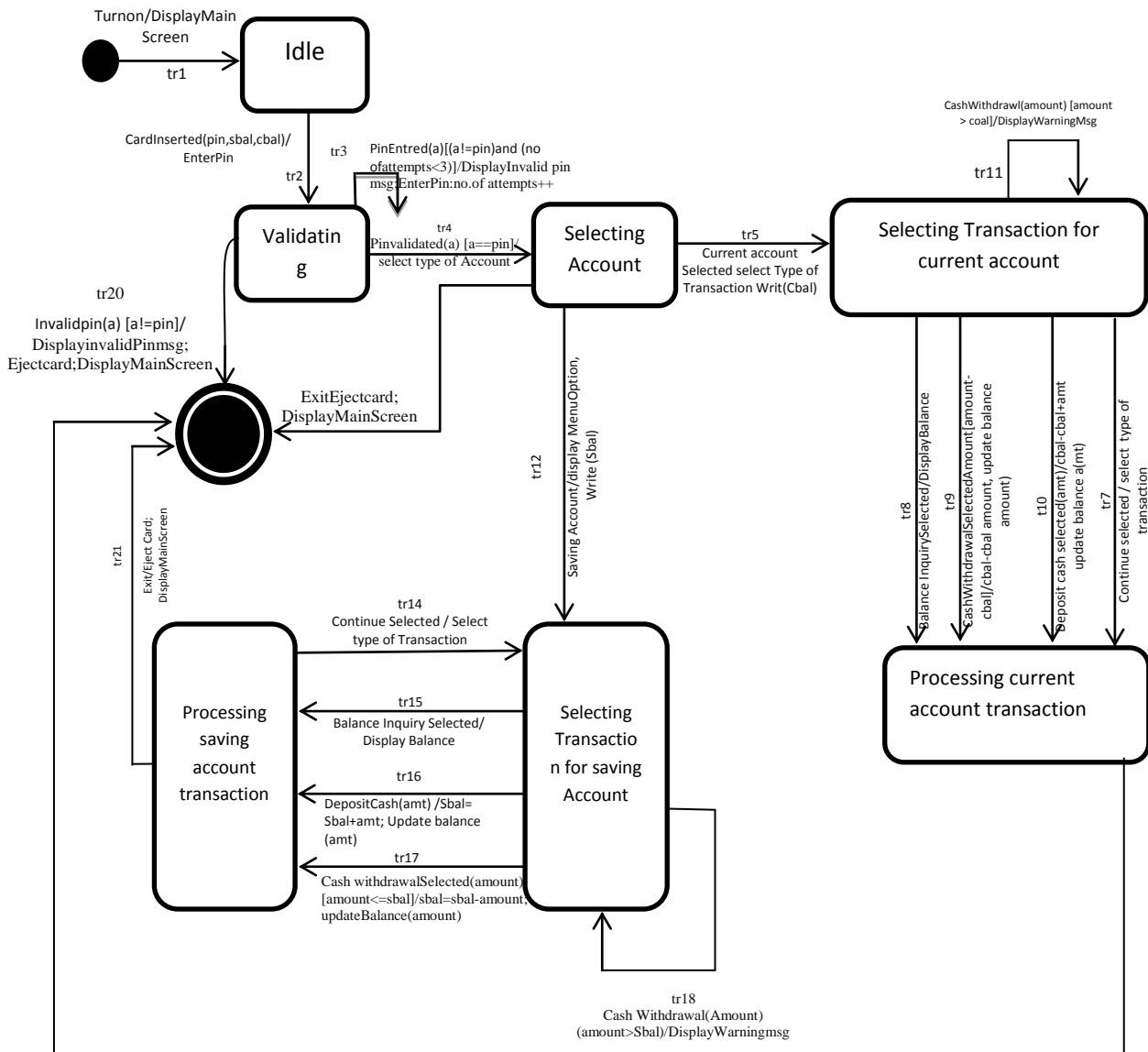


Fig. 31: State Chart Diagram of Modified ATM

## Identifying and Analysing State Chart Based Changes

A change will be considered as a state chart based change if it falls under any of the category being mentioned in the section 4.1.

After comparing the two state chart diagrams of original and modified version as shown in Fig.30 and Fig.31 respectively, we have encountered the changes illustrated as follows:

- i. Deleted Transitions:** The transition tr6 and tr13 have been deleted from the state chart diagram of the system. The transitions are deleted so as to meet the constraint that a user can have one account in the bank, there cannot be functionality in the system which allows user to do a transaction from one account and then continue to do a transaction from another account. Hence there is no need of the transition tr6 and tr13 and thus are removed from the state chart diagram.
- ii. Added State:** A new state named “Printing” is added, leading to addition of new transaction which is tr18, tr19, tr20, tr21, tr22. The state is added so as to meet the functionality of providing a print receipt of the transaction as it will be allow the user to know about the details of the transaction which he has committed. Due to addition of this state new transitions are also added, tr18 to allow a user to get the print receipt of a transactions with saving account, tr19 to allow a user to get the print receipt of the transaction with current account, tr20 to eject the card after the receipt has been printed and the transaction is completed, tr21 and tr22 to allow a user to continue with another transaction for the same account being previously selected for the previous transaction.

Therefore if we consider these added transitions as change, the change of addition of “printing” state will be automatically considered, hence there is no need to change consider it as a separate change.
- iii. Added Transitions:** The transition which have been added to the state chart diagram of the system are tr3, tr18, tr19, tr20, tr21, tr22. The transition tr3 is added to provide user authentication in a better way, by checking the

number of attempts a user has made with wrong pin. If the user has entered an incorrect pin, the system will display a message of invalid pin and will ask the user to enter the pin again and will verify it but even if for the third time user enter wrong pin then card is ejected with a message of invalid pin.

If the numbers of attempts are less than three and the pin is verified the user is allowed to continue to make a transaction. The transition tr18, tr19, tr20, tr21, tr22 are added with the addition of new state named printing and the reasons for which have been explained above.

- iv. **Changed transitions:** The transition which have been changed are tr16 and tr16. **In tr20** the guard condition has changed earlier the condition will be true if the pin being entered by the user is not equal to the actual pin but later the condition will satisfy if two sub conditions will be true i.e. the pin entered by the user must be equal to the actual pin and also the number of attempts a user has made with the wrong pin must be equal to three. The transition is changed so as to provide the functionality that a user can make only two attempts with a wrong pin, if for the third time a user does so the card will be ejected and a message of invalid pin will be displayed.

**In tr17** there is change in the action part of the transition. The transition corresponds to the cash withdrawal transaction from saving accounts, earlier the transaction was not deducting the amount withdrawn from the balance of the account, but as a change this functionality is provided so as to complete the transaction.

#### **Analysed set of state chart based changes**

After identifying and analysing changes in the state chart diagrams of original and modified system so that all the changes are considered and also no change is considered unnecessarily for testing we will get the analysed set of state chart based changes and it is named as (SC.1), which will be further used in the next steps of the Approach for the development of the test suite for regression testing.

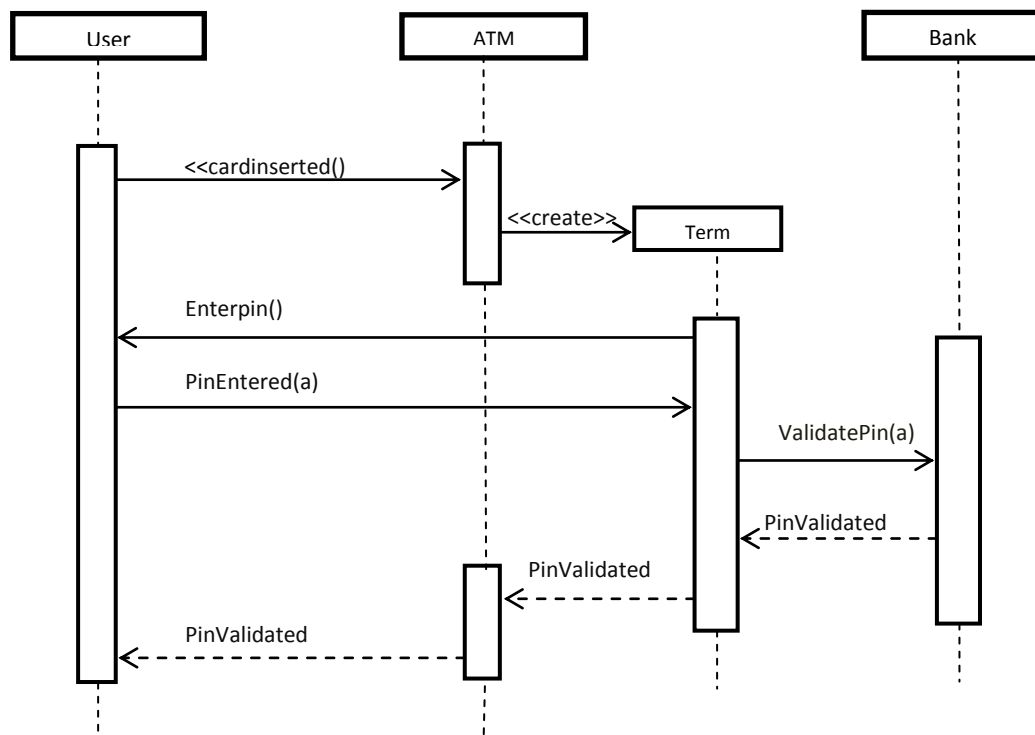


The next step is to identify the changes in sequence diagrams of the original and modified system by comparing the two sequence diagram and then analysing the changes to determine set of state chart based changes of the system being considered.

The sequence diagram for some scenarios such as validate pin, invalid pin, making a cash withdrawal transaction from savings account are considered to describe the sequence in which messages are exchanged between the different objects of the ATM system for each scenario.

The Fig. 32 shows the sequence diagram for scenario validate pin of the original version of the system, which remains same for modified version of the system as well. The sequence of message exchange is described as follows:

- i. User sends card is inserted message to ATM, resulting in creating an object of session type by ATM.
- ii. Session sends enter pin message to User, after which User sends pin entered to Session. Then Session sends validate pin message to Bank. Bank after verifying sends valid pin return message to session.
- iii. Session then after receiving valid pin message forwards it to ATM.
- iv. ATM upon receiving valid pin message forwards it to User to notify that the pin is validated, now the user can continue to use the system further.

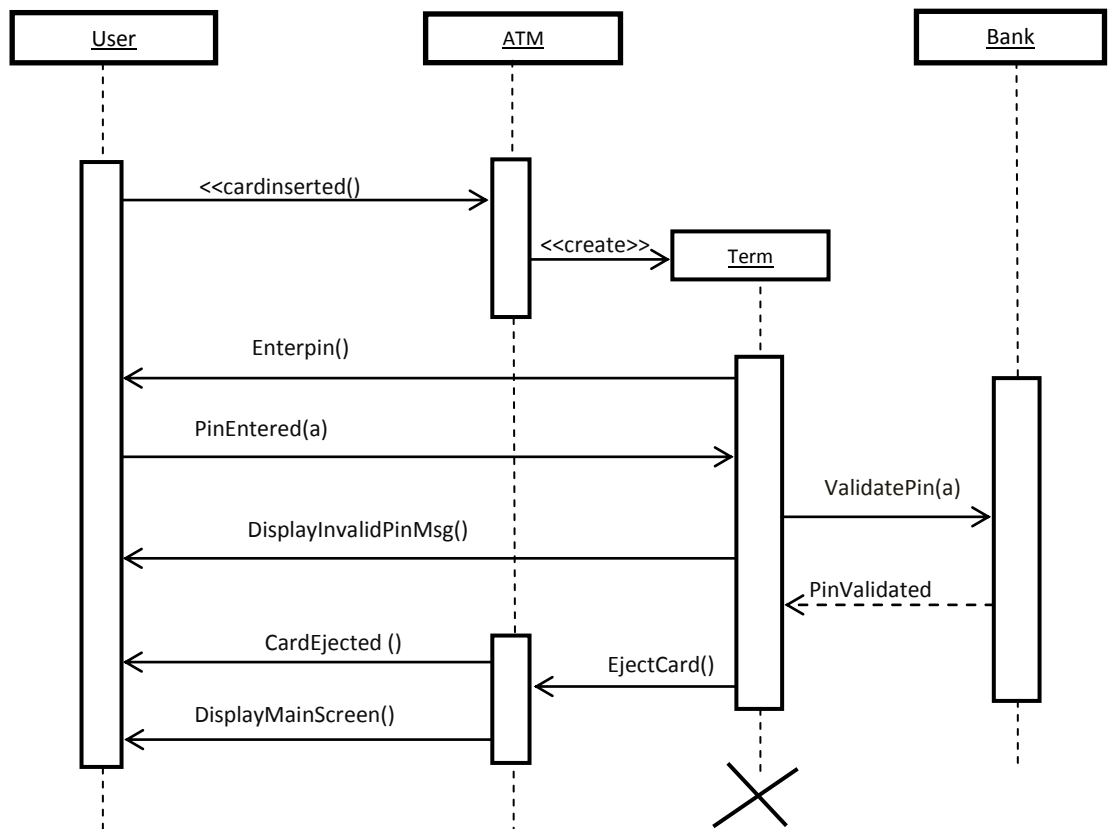


**Fig. 32: Sequence Diagram of Valid Pin of ATM System**

## INVALID PIN SCENARIO

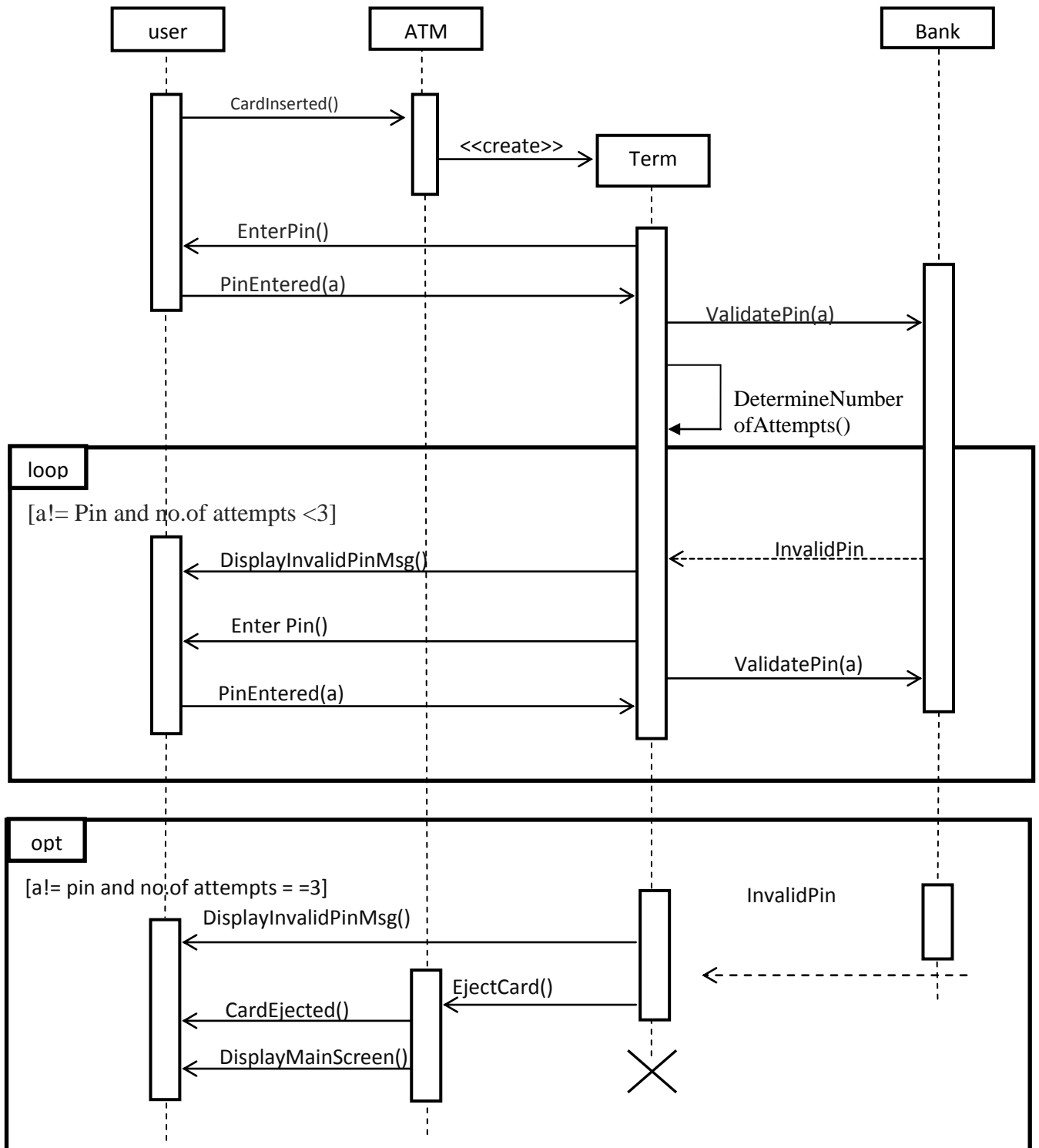
Fig. 33 shows the sequence diagram for scenario of invalid pin of the original version of the ATM system i.e. if the user enters a wrong pin then the sequence of message exchanged between the involved objects are described as follows:

- i. User sends card is inserted message to ATM, resulting in creating an object of Session type by ATM.
- ii. Session sends enter pin message to User, after which User sends pin entered to Session. Then Session sends validate pin message to Bank. Bank verifying sends valid pin return message to Session to notify that the user is not authenticated.
- iii. Session sends display invalid pin message to User to notify user that the pin is entered is incorrect and sends eject card message to ATM, so that ATM can eject out card.
- iv. ATM sends card ejected message and display main screen message to User.



**Fig. 33: Sequence Diagram of Invalid Pin of Original ATM System**

Fig. 34 shows the sequence diagram of modified system for scenario of invalid pin. To identify the changes, the two sequence diagram of original and modified system is compared. A change will be considered as a sequence diagram based change if it falls under any of the category being mentioned in the shown 4.1.



**Fig. 34: Sequence Diagram of Invalid Pin of Modified ATM System**

## Identifying and Analysing Sequence Diagram Based Changes for Invalid Pin Scenario

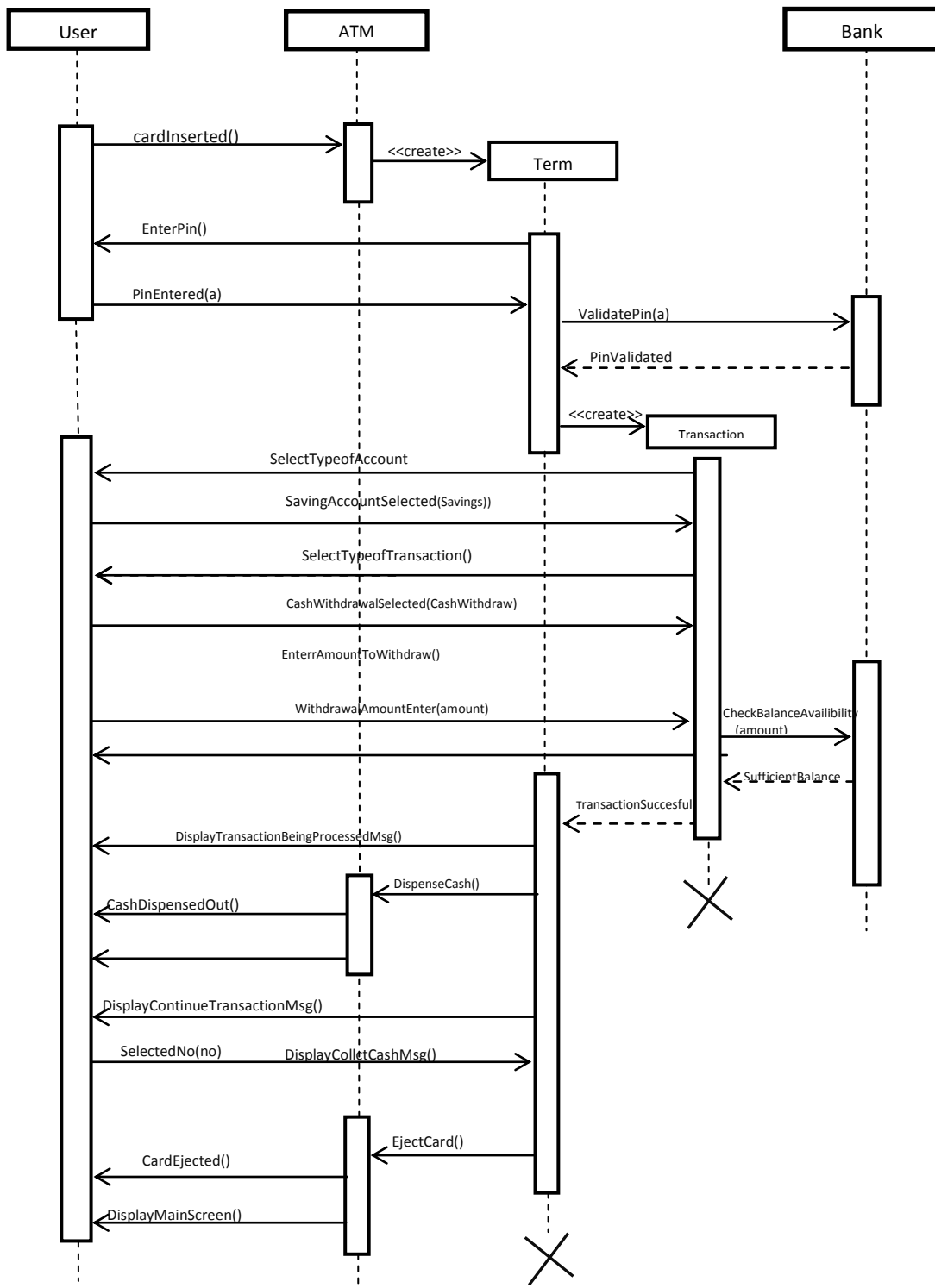
After comparing the two sequence diagram for invalid pin scenario of original and modified versions shown in Fig.33 and Fig.34 respectively, we have identifying and analysed the changes as follows:

- i. **Added Method:** A new method named “determine the number of attempts” is added in the sequence diagram of modified version of the system to keep a check on number of attempts a user has made. As the added combination fragments illustrated below, uses number of attempts in their guard condition hence if we are considering them as a change, there is no need to consider it a separate change.
  
- ii. **Added Combination Fragments – (loop, option):** A loop combination fragment with a guard condition checking the number of attempts is less than three and (a!= pin) condition i.e. the pin entered by the user is incorrect, is added and a option combination fragment with a guard condition checking the number of attempts are equal to three and invalid pin i.e. (a != pin) is also added. The change have been done to provide functionality to the user that if user enters a wrong pin, the system must ask user to enter again but if this happens for the third time the card must be ejected out and a message of invalid pin must be displayed.

## CASH WITHDRAWAL from SAVING ACCOUNT SCENARIO

Now considering the scenario of making a transaction to withdrawn cash from savings account, let us find out the changes which have occurred in the ATM system with respect to making cash withdrawal from savings account by comparing the sequence diagrams for this scenario of the original and modified system.

Fig. 35 shows the sequence diagram for scenario of making a transaction to withdraw cash from savings account for the original ATM system.

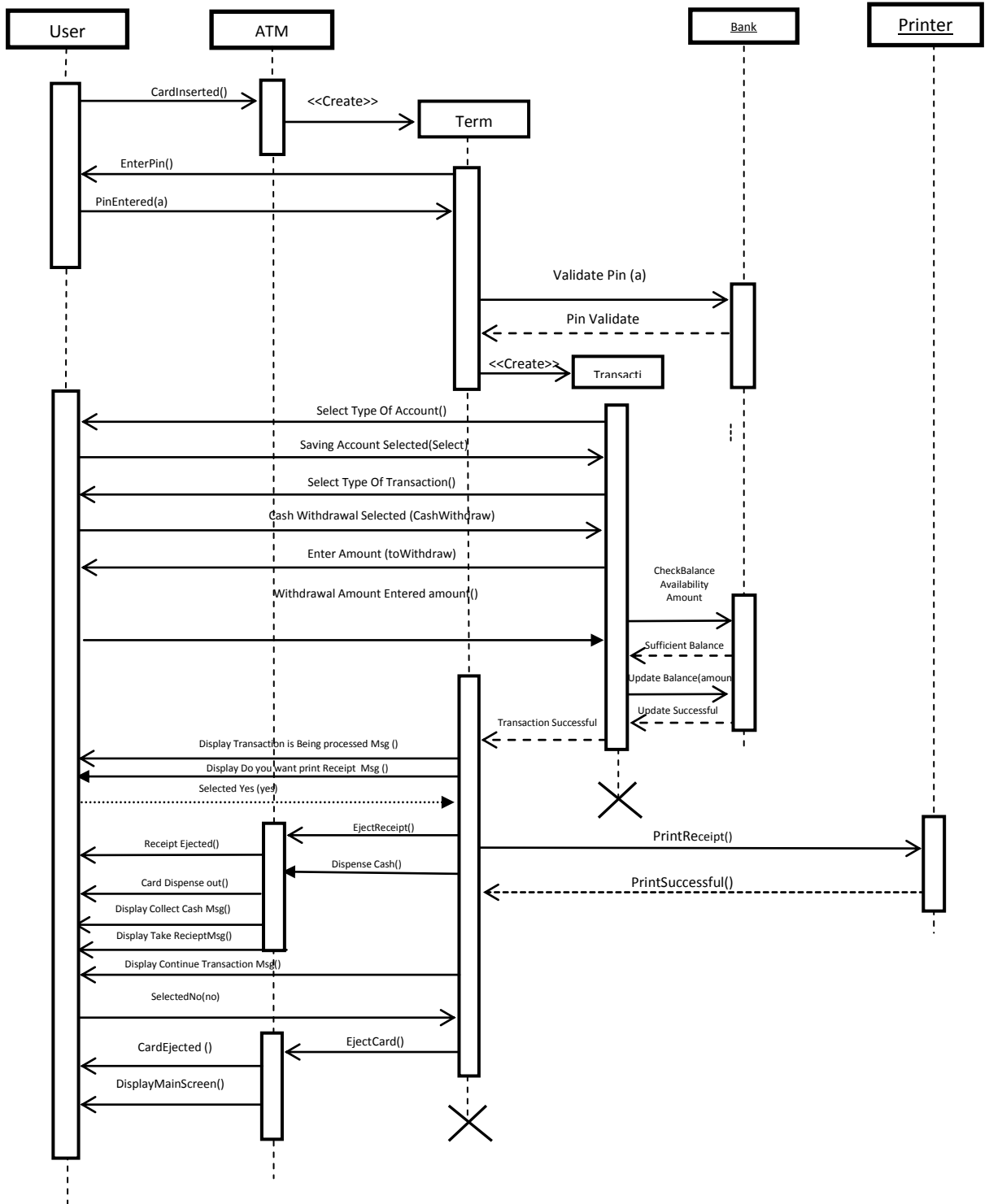


**Fig. 35: Sequence Diagram of Cash Withdrawal from Savings of Original ATM System**

The arrangement of messages exchanged between the involved objects for making a cash withdrawal transaction from savings account as shown in Fig. 35 are described as follows:

- i.** Firstly user authentication is done by validating the pin and the sequence of messages used for validating pin is same as described earlier with the sequence diagram of validate pin
- ii.** When the Session receives return message from Bank, it creates a new object of type Transaction. Then Transaction sends select type of account message to User, after which User sends saving account selected message to Transaction.
- iii.** Then Transaction sends message asking User to select type of transaction, after which User sends message to Transaction notifying that cash withdrawal is selected as type of transaction Then Transaction asks User to enter the amount to be withdrawn after which User sends message to Transaction that amount is entered.
- iv.** Then Transaction sends message to bank that check availability of balance, to which Bank sends return message notifying that there is sufficient balance. Upon receiving this message it sends transaction successful message to session.
- v.** Session then sends a message to User to notify that the transactions being processed and after then sends dispense cash message to ATM.
- vi.** ATM sends message to User that cash is dispensed out and request user to take cash. After which Session asks User that whether it wishes to continue, then if User sends message no, Session sends message to ATM to eject card, and the Session type object is destroyed.
- vii.** Finally ATM sends User message that card is ejected and display main screen.

Fig. 36 shows the sequence diagram for scenario of making a transaction to withdraw cash from savings account for the modified system.



**Fig. 36: Sequence Diagram of Cash Withdrawal from Savings of Modified ATM System**

## Identifying and Analysing Sequence Diagram Based Changes for Cash Withdrawal from Savings Account Scenario

In order to identify the sequence diagram based changes we need to compare the two sequence diagrams for making a cash withdrawal transaction from savings account for original and modified version of the ATM system as shown in Fig. 35 and Fig.36 respectively, and after comparing them we have encountered the following changes:

- i. **Added Methods:** the method named Update Balance (amount) is added so that when the user withdraws cash, the balance of the account is updated i.e. the amount is deducted from the balance. Also a return message Update Successful is added but as it is a result of addition of update balance (amount) method, hence there is no need to consider it as a separate change.

In order to allow a user to make a choice about he wants a print receipt or not for the transaction some methods are added such as a method named Display Do You Want Print Receipt Msg() is added to ask a user for the print receipt, a method named Selected Yes(yes) is added to allow a user to select yes if he needs a print receipt, then Print Receipt() method is added to issue a request to printer, as a result of which a return message Print Successful is added hence there is no need to consider it as a separate change, Eject Receipt() is added to indicate the system to eject out the receipt, Receipt Ejected () to indicate the user entity that receipt has been ejected out and Display Take Receipt Msg() to display a message requesting user to take out the receipt.

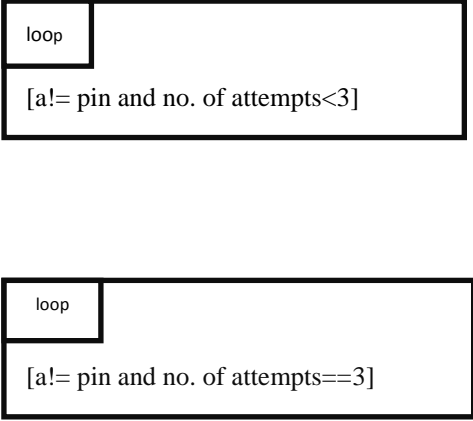
- ii. **Added Object:**A new object is added of type “Printer” so as to allow the user to get a print receipt of the transaction done, which has lead to addition of methods related with the print functionality being considered in the added methods change.

Therefore if we are considering above added methods as change, the change of addition of “Printer” type object will be automatically considered, hence there is no need to consider it as a separate change.

## Analysed Set of Sequence Diagram Based Changes for Cash Withdrawal from Savings Account Scenario and Invalid Pin Scenario

After identifying and analysing changes in the sequence diagrams of the scenarios invalid pin and making a cash withdrawal transaction from the original and modified system we have encountered following as final set of sequence diagram based changes. Table VI shows the set sequence diagram based changes for ATM system.

**TABLEVI: SEQUENCE DIAGRAM BASED CHANGES**

ADDED METHODS	ADDED COMBINATIONFRAGMENTS
<p>Update Balance(amount) →</p> <p>Display Do You Want Print ReceiptMsg() →</p> <p>Selected Yes(Yes) →</p> <p>Print Receipt() →</p> <p>Eject Receipt() →</p> <p>Receipt Ejected() →</p> <p>Display Take ReceiptMsg() →</p>	

### Step 1.3: Mapping the Analysed Sequence Diagram Based Changes with State Chart Based Changes

After the changes in the state chart diagrams of the system and the system and the changes in the sequence diagrams of the system have been identified and analysed, the two analysed sets of changes are mapped with each other to determine the associated state chart based change corresponding to the sequence diagram based changes.

i. **Mapping of Added Combination Fragments with State Chart Based Changes:**

Two combination fragments are added, loop and option now considering these changes, we will find out the corresponding changes in the state chart diagram by checking that which transitions of the modified state chart diagram consists of same guard condition which exists in the added option or loop combination fragment. After determining that we found out that the corresponding transitions are **tr3 and tr15**.

- a. Transition t3 contains guard condition i.e.  $((a! = \text{pin}) \text{ and } (\text{no. of attempts} < 3))$  i.e. the pin which user entered doesn't match the actual pin and number of attempts user has made are less than three, which is same as of guard condition of loop combination fragment also i.e.  $((a! = \text{pin})) \text{ and } (\text{no. of attempts} < 3)$ , therefore the change in sequence diagram of loop combination fragment corresponds to transition tr3 of changed state chart diagram.
- b. Transition tr15 modified, has guard condition  $((a! = \text{pin}) \text{ and } (\text{no. of attempts} = 3))$  which is same as that of the guard condition of option combination fragment in the changed sequence diagram i.e.  $((a! = \text{pin}) \text{ and } (\text{no. of attempts} = 3))$ , therefore the change in sequence diagram of option combination fragment corresponds to transition tr15 of the changed state chart diagram.

ii. **Mapping Added Methods Change with State Chart Based Changes:**

Now considering the change of all added methods, we will find out the corresponding changes in the state chart diagram by checking that which transitions of the modified state chart diagram consists of these methods. After checking that we found out that the corresponding transitions are tr17, tr18 and tr19.

- a. Update Balance (amount) methods corresponds to transition tr17 as he action part of transition rt17 consists of this method.
- b. Other added methods related with the print functionality such as Print Receipt( ), Eject Receipt( ) and other listed in the analysed set of sequence based changed corresponds to transitions tr18 and tr19 which also provide a print receipt functionality, as their event and action part consists of these added methods.

## Mapped Sequence Diagram Based Changes with State Chart Based Changes

The Table VII shows the analysed sequence diagram based changes and associated state chart based changes.

**TABLEVII: MAPPED SEQUENCE DIAGRAM BASED CHANGESWITH STATE CHART BASED CHANGES**

<b>Sequence Diagram Based Change</b>	<b>Associated State Chart Based Changes(SC.2)</b>
Added Combination Fragments (loop, option)	Added Transition tr3, Changed Transition tr16
Added Methods Change	Added Transition tr20,tr21

### Step 1.4: Determine Final Set of Changes

Now when the sequence diagram based changes are mapped with the state chart based changes to get a set of associated state chart based changes(SC.2) we will take union of SC2 and analysed state chart based changes(SC.1) to determine final set of changes. After taking the union of the two sets of changes SC.1 and SC.2 we get the final changes as shown in Table VIII.

**TABLE VIII: FINAL SET OF CHANGES FOR ATM SYSTEM**

<b>Deleted Transitions</b>	<b>t6 and t13</b>
<b>Added Transitions</b>	<b>tr3,tr18,tr19,tr20,tr21 and tr22</b>
<b>Changed Transitions</b>	<b>Tr14 and tr16</b>

### Step 2: Regression Test Suite Development

Now when we have identified the final set of changes, next step is development of regression test suite to test the changes being identified. We have the original test suite(Test Suite 1) to test the features of the original ATM system. The Test Suite has three fields i.e. test case Id – a unique ID for each test case. Purpose which states the functionality of test case i.e. which features of the system is tested by the particular

test case and transitions associated which describes the involved transitions of the original state chart diagram of the ATM in executing the particular test case. Table IX shows the Test Suite 1 as follows:

**TABLE IX: ORIGINAL TEST SUITE 1 FOR ATM SYSTEM**

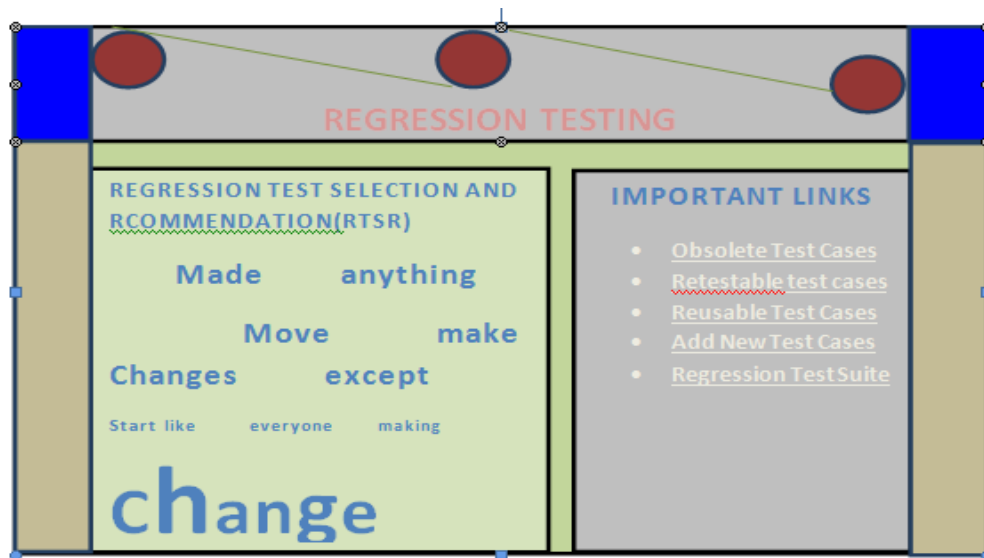
Test Case Id	Purpose	Transitions Related
Test1	For <i>failed test user authentication</i>	tr1 tr2
Test2	For testing <i>immediate exit after authentication</i>	tr1 tr2 tr4
Test3	For test <i>cash withdrawal from current account</i>	tr1 tr2 tr4 tr5 tr9
Test4	For test <i>cash deposit in current account</i>	tr1 tr2 tr4 tr5 t10
Test5	For test <i>balance inquiry from current account</i>	tr1 tr2 tr4 tr5 tr8
Test6	For test <i>cash withdrawal from savings account</i>	tr1 tr2 tr4 tr12
Test7	For test <i>cash deposit in saving account</i>	tr1 tr2 tr4 tr12 tr16
Test8	For test <i>balance enquiry from saving account</i>	tr1 tr2 tr4 tr12 tr15
Test9	For test continue another <i>transaction from same Account after cash withdrawal from savings</i>	tr1 tr2 tr4 tr12 tr14
Test10	For test continue another <i>transaction from same Account after cash deposit from savings</i>	tr1 tr2 tr4 tr12 tr16 tr14
Test11	For test continue another <i>transaction from same Account after balance inquiry</i>	tr1 tr2 tr4 tr12 tr15 tr14

	from <i>savings</i>	
Test12	For test continue another <i>transaction</i> from same Account after <i>cash withdrawal</i> from <i>current</i>	tr1 tr2 tr4 tr5 tr9 tr7
Test13	To test continue another <i>transaction</i> after cash <i>Deposit</i> from <i>current</i>	tr1 tr2 tr4 tr5 tr10 tr7
Test14	For test continue another <i>transaction</i> from same Account after balance inquiry from <i>current</i>	tr1 tr2 tr4 tr5 tr8 tr7
Test15	For test a <i>successful authentication</i>	tr1 tr2 tr4
Test16	For test <i>cash withdrawal</i> with <i>saving account</i> Then continue with <i>different account</i>	tr1 tr2 tr4 t12 tr17 tr14 tr13

In order to validate the approach Regression Test Selection and Recommendation (RTSR), an application is designed which provides the following features:

- i. Classification of the original test suite (Test Suite 1) as illustrated above, by determining the obsolete test cases, retestable test cases and reusable test cases.
- ii. Recommendation of new test cases by allowing user to add a new test case, when the original test suite does not contain test case to test a newly added feature of the system.
- iii. To determine the regression test suite consisting of retestable test cases and newly added test cases (if any).

The Fig. 37 shows the home page of the application designed for validating for approach Regression Test Selection and Recommendation (RTSR), providing option to determine the obsolete test cases, retestable test cases, reusable test cases, add a new test case and determine the regression test suite.



**Fig. 37: Home Page**

### **Step 2.1: Classification of Original Test Suite**

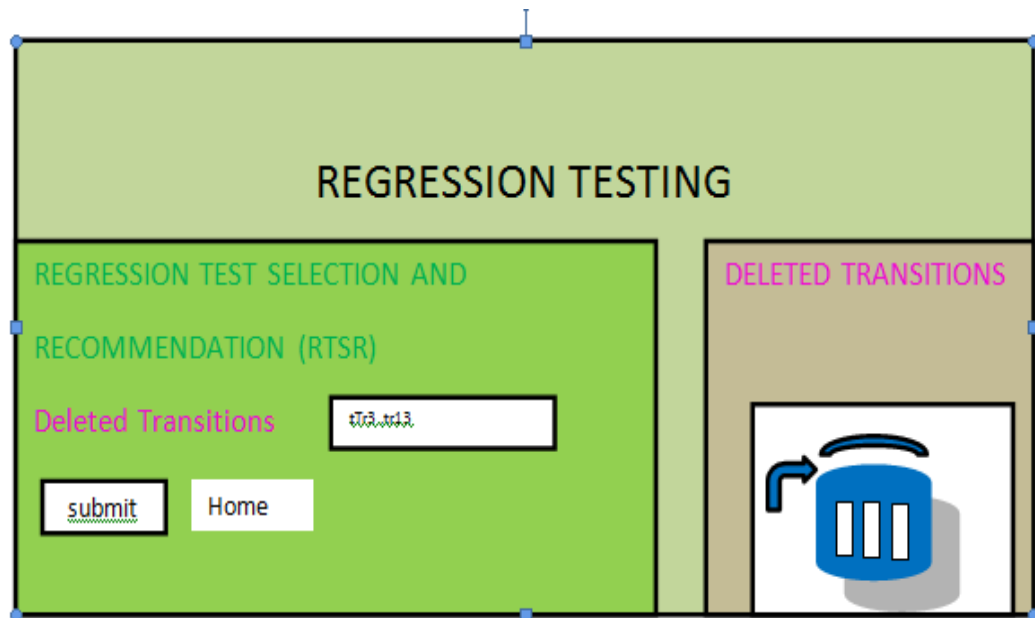
Now when we have identified the final set of changes, we will classify the original test suite into three categories i. e. retestable test cases, obsolete test cases and reusable test cases[29] by using the change information found out the step 1.

#### **Step 2.1.1: To Determine Obsolete Test Cases.**

The test cases which are not required to be executed to test the changed system are obsolete test cases, the test cases which contain at least one of the deleted transitions in their associated transitions will be selected and are a categorized as obsolete test cases.

In order to determine obsolete test cases, deleted transitions- tr6, tr13 being determined in step 1 in the final set of changes listed in Table VIII have to be entered.

Fig. 38 shows the page for entering deleted transitions and once deleted and once submit button is clicked, page will be redirected to the page displaying obsolete test cases.



**Fig. 38: Deleted Transitions**

Fig. 39 shows the page displaying the obsolete test cases, these test cases are no more valid for the modified system as they correspond to those parts of the system which no more exists and therefore will not form the part of regression test suite.

**REGRESSION TESTING**

**REGRESSION TEST SELECTION AND RECOMMENDATION (RTSR)**  
**OBSOLETE TEST CASES**

TEST ID'S	PURPOSE
tr7	To test case deposit in saving account
tr8	To test balance enquiry from saving account
tr9	To test continue another transaction from same account after cash withdrawal from savings
tr10	To test continue another transaction from same account after cash deposit from savings
tr11	To test continue another transaction from same account after balance inquiry from savings
tr12	To test continue another transaction from same account after cash withdrawal from current

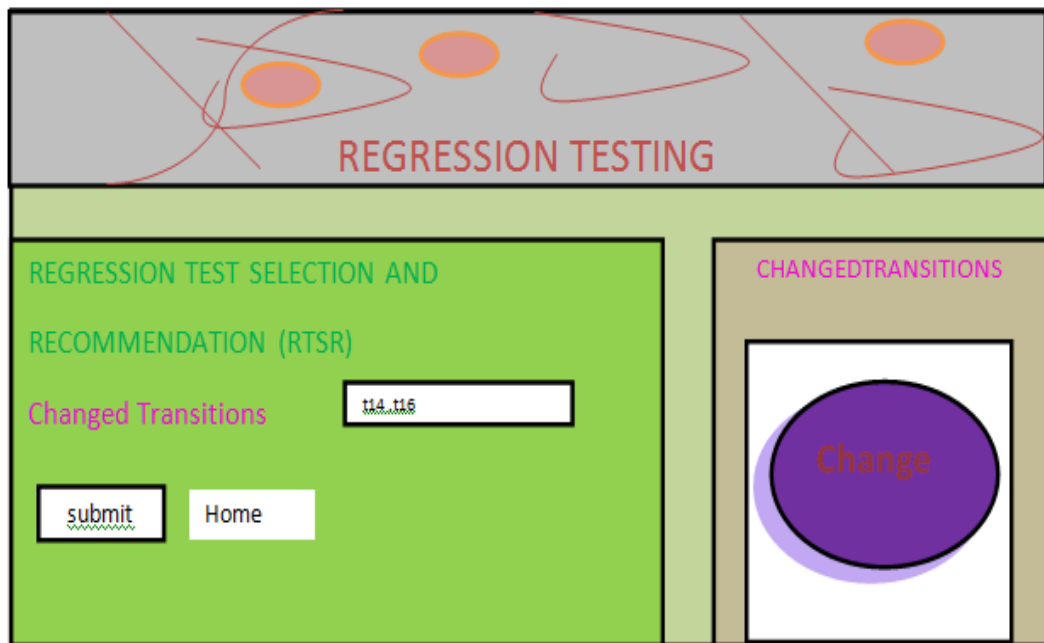
**NO MORE VALID!!**

**OBSOLETE**

**Fig. 39: Obsolete Test Cases**

### Step 2.1.2: To Determine Retestable Test Cases

In order to determine retestable test cases, we have to enter the changed transition tr20, tr17 being listed in Table VIII in final set of changes determined in step 1. Fig. 40 shows the page for entering the changed transitions, where after entering the changed transitions if we click submit it will redirect us to page displaying retestable test cases.




**Fig. 40: Changed Transitions**

Fig. 41 shows the page displaying the retestable test cases; the test cases which contain at least one of the changed transitions in their associated transitions will be selected and are categorized as retestable test cases.

REGRESSION TESTING	
REGRESSION TEST SELECTION AND RECOMMENDATION (RTSR)	
RETESTABLE TEST CASES	
TEST ID'S	PURPOSE
T1	To Test failed user authentication
T6	To Test case withdrawal from saving account
T9	To Test continue another transaction from same account after cash withdrawal from savings

TEST AGAIN!!



**Fig. 41: Retestable Test Cases**

### Step 2.1.3: To Determine Reusable Test Case

Fig. 42 shows the page displaying the reusable test cases. The test cases which can be reused are reusable test cases as these test cases are related with the component of the system which have not been changed or modified. All the test cases in the initial test suite except obsolete test cases and retestable test cases are reusable test cases. As these test cases do not correspond to any change in the system, they are not required to be executed again hence are not included in regression test suite but are still valid for the new system.

REUSABLE TEST CASES	
TEST ID	PURPOSE
T2	To test an immediate exit after
T3	To test cash withdrawal from current account
T4	To test cash deposit in current account
T5	To test balance inquiry from current account
T7	To test cash deposit in saving account
T8	To test balance enquiry from saving account
T10	To test continue another transaction from same account after cash deposit from savings
T11	To test continue another transaction from same account after bal. enquiry from
T12	To test continue another transaction from same account after cash withdrawal from current
T13	To test continue another transaction after cash deposit from current
T14	To test continue another transaction from same account after bal. inquiry from current
T15	To test a successful authentication

Reusable!!




Fig. 42: Reusable Test Cases

### Step 2.2 Addition of new test cases(if required)

As there may be a possibility that some component or some functionality is added in the system and the original test suite (Test Suite1) does not contain any test case to test case to test that added functionality or added component, then he need arises to create new test cases, which will be included in regression test suite. Considering the change in the ATM system, the transitions which correspond to add functionality or added component **i.e. added transitions are tr3, tr17, tr18, tr19, tr20 and tr21** as

listed in TABLE VIII. To test these changes, we will create new test cases which will be added into regression test suite. The new test cases are illustrated as follows:

- i. **Transition tr3:** corresponding to transition tr3, a new test case is added, the purpose of which is to test the features that if user enters invalid pin and the number of attempts made with wrong pin are less than three, then user is asked to enter pin again and if verified the user is allowed to continue further to use system.

Test Case ID	Purpose	Transition Associated
Test 17	To enter invalid pin less than three times	tr1, tr2, tr3, tr4.

- ii. **Transitions tr15, tr16:** Corresponding to transitions tr15, tr16 a new test case added the purpose of which is to test the features of getting a print receipt for a transaction with savings account.

Test Case ID	Purpose	Transition Associated
Test18	To test getting a print receipt for a balance enquiry transaction with savings Account.	tr1, tr2, tr4, tr12, tr15, tr16

- iii. **Transition tr19, tr20:** Corresponding to transition tr19, tr20 a new test case added the purpose of which is to test the feature of getting a print receipt for transaction with current account.

Test Case ID	Purpose	Transition Associated
Test19	To test getting a print receipt for a balance enquiry transaction with current account	tr1, tr2, tr4, tr5, tr8, tr19, tr20

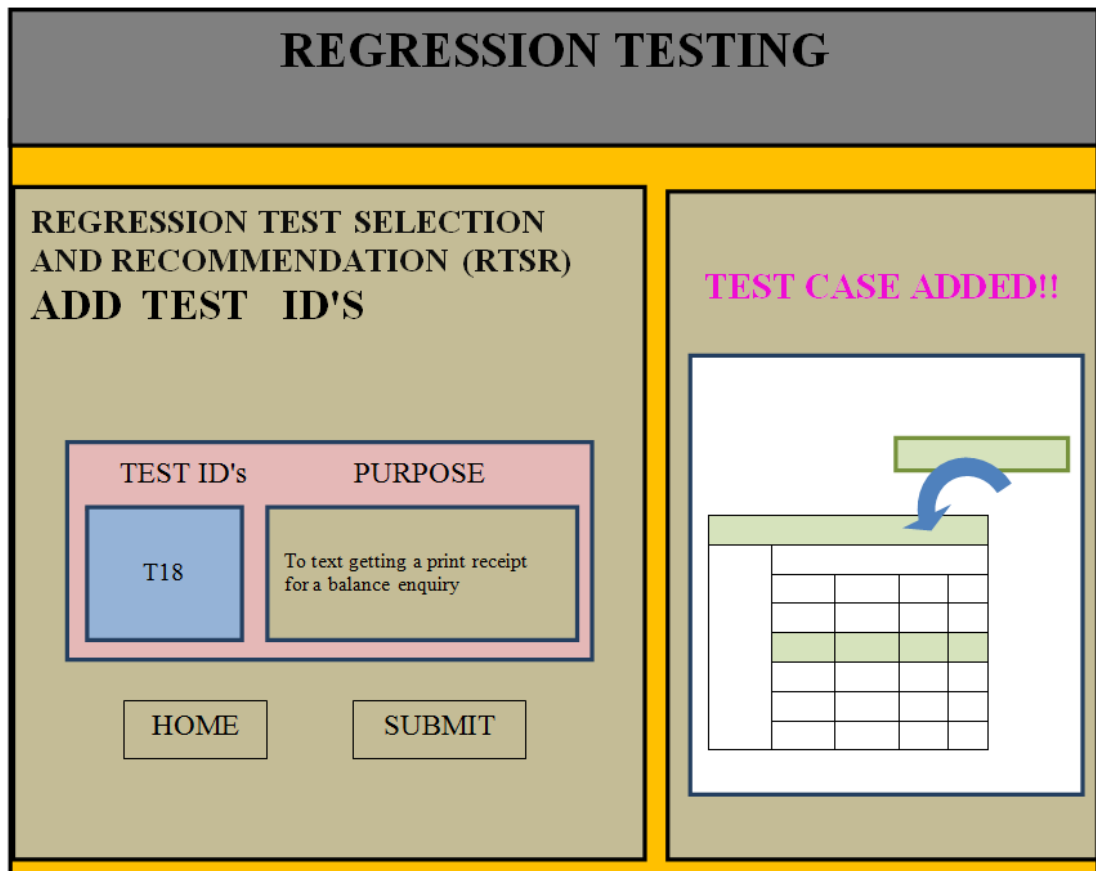
- iv. **Transition tr21:** Corresponding to transition tr21 a new test case added the purpose of which is to test the features of getting a print receipt for a transaction with current account and continue to make another transaction from same account.

Test Case ID	Purpose	Transition Associated
Test20	To test getting a print receipt for a balance enquiry transaction followed by cash withdrawal with same current account	t1, t2, t4, t5, t8, tr19, tr21, tr9, tr17

- v. **Transition tr22:** corresponding to transition tr22 a new test case added the purpose of which is to test the features of getting a print receipt for a transaction with savings account and continue to make another transaction from same account

Test Case ID	Purpose	Transition Associated
Test21	To test getting a print receipt for balance enquiry transaction followed by cash deposit with same savings account	tr1, tr2, tr4, tr12, tr15, tr18, tr22, tr16,

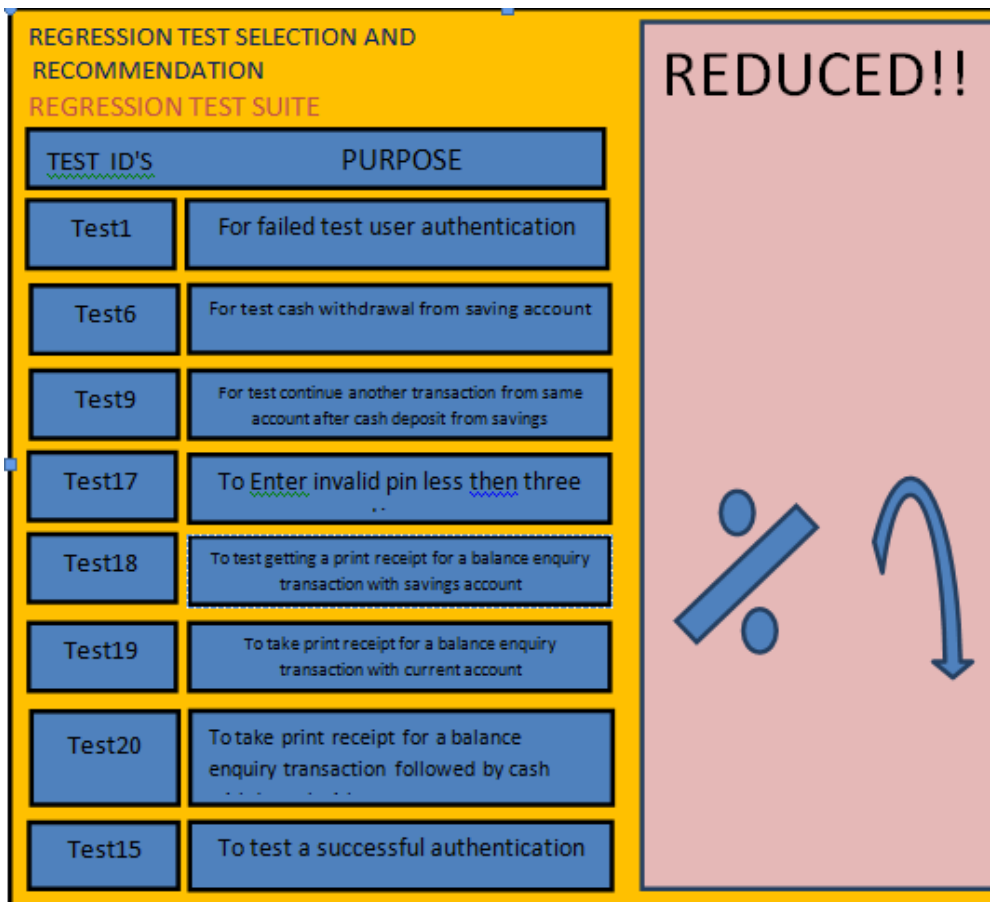
Fig. 43 shows the page to add a test case, with Test Case ID T23 and purpose to test getting a print receipt for a balance enquiry transaction with savings account



**Fig. 43: Add Test Case**

### **Step 2.3 Determination of Regression Test Suite**

Fig. 44 shows the page displaying the final reduced regression test suite as compared to original test suite. The test suite for regression testing must contain only those test cases which test the change in the system so as to perform an efficient regression testing i.e. use less time and resources and at same the time also reveals all the faults. The regression test suite will consist of retestable test cases and newly added test cases.



**Fig. 44: Regression Test Suite**

### 5.3 RESULTS

After applying the proposed approach – RTSR using the case study of Automated Teller Machine, we have got the following results:

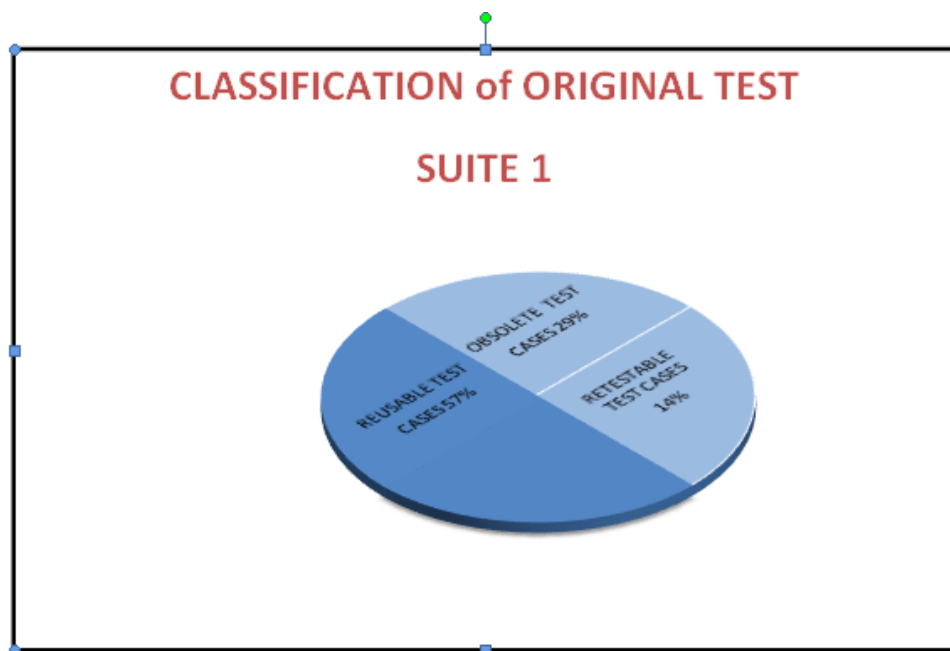
**i. Classification of Original Test Suite**

The approach RTSR has classified the original test suite into three categories; Table X shows the no. of test cases in each category.

**TABLEX : CLASSIFICATION OF ORIGINAL TEST SUITE**

<b>OBSOLETE TEST CASES</b>	<b>RETESTABLE TEST CASES</b>	<b>REUSEABLE TEST CASES</b>
<b>6</b>	<b>3</b>	<b>12</b>

Fig.45 shows the pie chart for classification of original test suite showing the percentage of each category of test cases, obsolete test cases forms 29% of the original test suite, retestable test cases forms 14% of the original test suite and reusable test cases forms 57% of the original test suite, therefore it can inferred that a large part of original test suite is reusable and only 14 per cent of the test cases are required to be tested again, which will results introducing the time and cost of testing.



**Fig. 45: Classification of Original Test Suite**

**ii. Recommendation of New Test Cases**

The approach RTSR has recommended five new test cases to test the new features being added to the ATM system. The test cases being recommended are illustrated in the previous section.

**iii. Reduction in Regression test Suite**

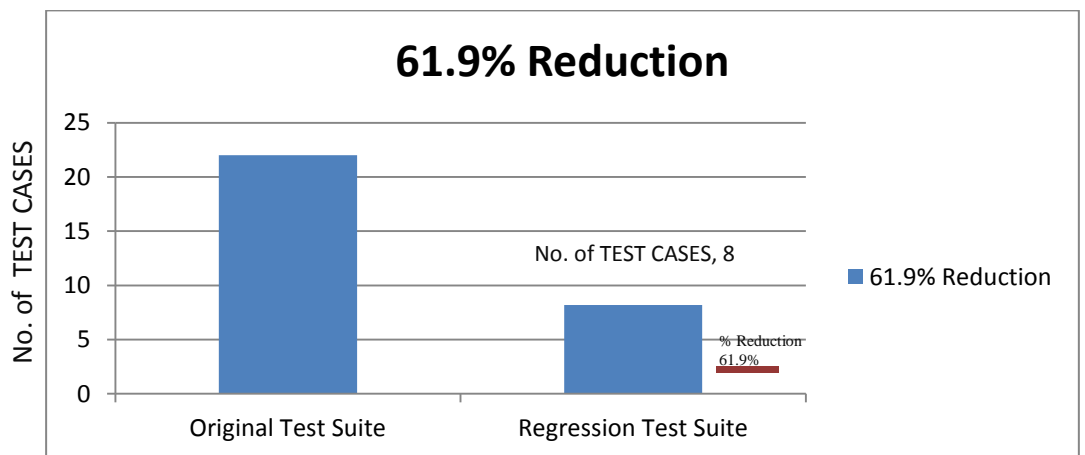
The approach RTSR has resulting in producing a regression test suite consisting of eight test cases, therefore the percentage reduction in regression test suite is 61.9%, which can be calculated using formula:

$\% \text{ Reduction} = (\text{No. of test cases in Original Test Suite} - \text{No. of test cases in Regression Test Suite}) * 100 / \text{No. of test cases in Original Test Suite}$

$$\% \text{Reduction} = (21-8)*100/21$$

$$\% \text{Reduction} = 61.9\%$$

Fig.46 shows the bar graph describing the number of test cases in original test suite and regression test suite being produced by using RTSR approach, also shows the percentage reduction in the regression test suite.



**Fig. 46: Percentage Reduction**

#### 6.1 Conclusion

The objectives set in chapter 3 have been fulfilled successfully. The approach proposed RTSR is successfully validated and does appropriate selection and recommendation of test cases. The experimental results indicate that RTSR selects lower number of test cases than original test suite and a percentage reduction of 61.9 % is achieved in regression test suite also all the changes are tested successfully proving that the approach is efficient. The approach assumes strong correspondence between state chart and sequence diagram, hence will be more efficient if there is strong correspondence between state chart and sequence diagram of the system being considered.

#### 6.2 Contribution

A new approach RTSR based on UML to perform regression testing of component based software using UML is proposed. The approach uses UML state chart diagrams and sequence diagram to identify the changes in the system. The change information collected is used to select the test cases for regression testing from the initial test suite and recommend new test cases if required to form regression test suite. An application is designed to validate the approach using the case study of automated teller machine and has successfully resulted in a regression test suite for the system being considered, having less number of test cases as compared to original test suite. After the results are analysed it can be inferred that a percentage reduction of 61.9% is attained.

### **6.3 Future Scope**

The future work aims to develop a fully automated tool which can identify the changes by comparing the state chart and sequence diagrams of the original and modified system and produce a reduced and effective regression test suite.

**Youtube video presentation link:**<https://youtu.be/3ugXwJ8VjjM>

## References

---

- [1] Roger S. Pressman, "Software Engineering: A Practitioner's Approach," 5<sup>th</sup> ed. New York: McGraw-Hill, 2007.
- [2] X.Cai, M.R. Lyu, K.F. Wong and R.Ko, "Component-based Software Engineering: Technologies, Development Frameworks, and Quality Assurance Schemes," In Software Engineering Conference, 2000. APSEC 2000.Proceedings.Seventh Asia – Pacific, pp. 372-379, IEEE, 2000.
- [3] J.Gao, H.- S. Tsao and Y. Wu, "Testing and Quality Assurance for Component Based Software," London :Artech House, 2003.
- [4] IEEEStd 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology IEEE Standards Board, 28 Sept 1990.
- [5] "SWEBOK: Guide to the Software Engineering Body of Knowledge," 2004 Edition, IEEE.[online] <http://www.swebok.org>, Accessed Thu 28 Jun 20007.
- [6] J.M. Voas and K.W. Miller, "Software Testability: The New Verification," IEEE Software, vol.12, no. 3, pp. 17-28, May 1995.
- [7] W.Zheng, "Applying Test by Contract to Improve Software Component Testability," Technical Report CIIPS\_ISERG\_TR-2007-02 Centre for Intelligent Information Processing System, School of Electrical Electronic and Computer Engineering, University of Western Australia, WA, Australia, 2007.
- [8] R.S. Freedom "Testability of Software Components," IEEE Transactions on Software Engineering, vol. 17, no. 6, pp. 553-564, June 1991.
- [9] R.V. Binder," Design for Testability in Object Oriented System," Communication of ACM, vol. 37, no. 9, pp. 87-101, Sep 1994, ACM Press.
- [10] J.Gao and M. Shih, "A Component Testability Model for Verification and Measurement," Proc. 29<sup>th</sup> Annual Intl on Computer Software and Applications Conf(COMPSAC 2005), Edinburgh, Scotland, 26-28 July 2005, IEEE Computer Society Press, 2005, pp.211-218.

- [11] G. Duggal and B.Surihttp.(2008). Understanding Regression Testing Techniques [Online].Available://www.rimtengg.com/coit2008/proceedings/SW15.pdf.
- [12] Harrold, M.Jean, J.A. Jones, T.Li, D. Liang, A.Orso, M.Pennings, S.Sinha, S. A. Spoon, and A.Gujarathi. "Regression Test Selection for Java Software."In ACM SIGPLAN Notices, vol. 36, no. 11, pp. 312-326.ACM, 2001.
- [13] A.Orso, M.J. Harrold, M. L. Soffa , H.Do, D. Rosenblum and G. Rothermel, "Using Component Meta-content to Support the Regression Testing of Component-Based Software", IEEE International Conference on Software Maintenance, pp.716-725,2001.
- [14] Y. Chen, D.Rosenblum, and K. Vo. TestTube: A System for Selective Regression Testing. In 16<sup>th</sup> Int'l. Conf. Softw. Eng.,pages 211-222, May 1994.
- [15] G. Rothermel and M.Harrold, "A Safe, Efficient Regression Test Selection Technique," ACM Trans. on Softw. Eng and Meth., 6(2):173-210, April 1997.
- [16] G. Rothermel, M.J. Harrold, and J. Dedhia, "Regression Test Selection for c++ Software," J. Softw. Testing ,Verif., and Rel., 10(2), June 2000.
- [17] L. White and H.Leung," A Firewall Concept for both Control-flow and Data-flow in Regression Integration Testing," In Conf. Softw. Maint., pages 262-270, November 1992.
- [18] T. Ostrand and M. Balcer, "The category-partition method for specifying and generating functional tests," Comm. ACM , 31(61), June 1998.
- [19] C.Mao and L. Yansheng," Regression Testing for Component-Based Software System by Enhancing Change Information," In Software Engineering Conference, 2005. APSEC'05. 12th Asia-Pacific,pp. 8-pp.IEEE, 2005
- [20] Muccini, Henry , M.S. Dias and D.J. Richardson, "Towards Software Architecture-Based Regression Testing." In ACM SIGSOFT Software Engineering Notes, vol. 30, no. 4, pp. 1-7. ACM, 2005.
- [21] M. S. Dias, Vieira and D.J.Richardson ,"Analyzing Software Architecture with Argus-I, "In Proc of the 22<sup>nd</sup> International Conference on Software Engineering (ICSE 2000), Limerick, Ireland, June 2000, pp.758-761.

- [22] H. Muccini, A. Bertolino, and P. Inverardi, "Using Software Architecture for Code Testing," In IEEE Transactions on Software Engineering, vol. 30, n. 3, Mar 2004, pp. 160-171.
- [23] Formal Methods for Software Architectures. Tutorial book on Software Architectures and Formal Methods. In SFM-03:SA Lectures, Eds. M. Bernardo and P. Inverardi, LNCS 2804,2003.
- [24] J.Gao, D. Gopinathan& Q. M.J. He, "A Systematic Regression Testing Method & Tool for Software Components," 30<sup>th</sup> Annual International Conference on Computer Software & Applications, Vol. 1.pp. 455-466, 2006.
- [25] Marini, Leonardo, S. Papagiannaskis, and M.Pezze, "Compatibility and regression testing of COTS-component-based software," In Software Engineering, 2007. ICSE 2007. 29<sup>th</sup> International Conference on ,pp. 85-95. IEEE, 2007.
- [26] Marini, Leonardo, and M.Pezze," Behavior capture and test: Automated analysis of component integration," In Engineering of Complex Computer Systems, 2005. ICECCS 2005.Proceedings. 10<sup>th</sup> IEEE International Conference on, pp. 292-301. IEEE, 2005
- [27] C.Mao," Built-in –Regression Testing for Component-based Software Systems."In 31<sup>st</sup> Annual International Conference on Computer Software & Applications, Vol.2, pp. 723-728, 2007.
- [28] Q. Farooq, M. Iqbal , Z. Malik, and A. Nadeem, " An Approach for Selective State Machine based Regression Testing," In Proceedings of the 3<sup>rd</sup> International workshop on Advances in model based testing, pages 44-52. ACM, 2007.
- [29] H.K.N. Leung, L. White," Insights into Regression Testing Software Testing," In Proceedings of Conference on Software Maintenance. On page(s): 60-69, ISBN: 0-8186-1965-1, Oct 1989.
- [30] L. Naslavsky and D. Richardson, "Using Traceability to Support Modal-Based Regression Testing," In Proceedings of the twenty-second IEEE/ACM International conference on Automated software engineering, ASE '07, Pages 567-570. ACM, November 2007.

- [31] A. Ali, A. Nadeem, Z. Iqbal, and M. Usman, "Regression Testing Based on UML Design Models," In Proceedings of the 13<sup>th</sup> Pacific Rim International Symposium on Dependable Computing, pages 85-88, 2007.
- [32] R. Gorthi, A. Pasala, K. Chanduka, and B. Leong, "Specification-based Approach to select Regression Test Suite to Validate Changed Software," In Proceedings of the 2008 15<sup>th</sup> Asia –Pacific Software Engineering Conference, pages 153-160, 2008.
- [33] Q. Farooq, M. Iqbal, Z. Malik, and M. Riebisch, "A Model-based Regression Testing Approach for Evolving Software Systems with Flexible Tool Support," In 17<sup>th</sup> IEEE International Conference on Engineering of Computer-Based Systems (ECBS), pages 41-49. IEEE Computer Society, March 2010.
- [34] Cibulski, Hagai, and A. Yehudai, "Regression Test Selection Techniques for Test-Driven Development." In Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on, pp. 115-124. IEEE, 2011.
- [35] C. Tao, B. Li and J. Gao, "Regression Testing of Component-Based Software: A Systematic Practice Based on State Testing," 13<sup>th</sup> IEEE International Symposium on High-Assurance Systems Engineering, pp. 29-32-, 2011.
- [36] T. Sen and R. Mall, "State-Modal-Based Regression Test Reduction for Component – Based Software," ISRN Software Engineering , vol. 2012, Article ID 561502, 9 pages, 2012.
- [37] Ural, Hasan and H. Yenigun, "Regression Test Suite Selection using Dependence Analysis," Journal of Software : Evolution and Process 25, no. 7 (2013): 681-709.
- [38] Yang, Jinlin, D. Evans, Deepali Bhardwaj, T. Bhat, and M. Das, "Perracotta: Mining Temporal API Rules from Imperfect Traces," In Proceedings of the 28<sup>th</sup> International Conference on Software Engineering, pp. 282- 291. ACM, 2006.