

# Combining UML Interaction Diagrams and State-Charts for Testing of Object Oriented Software Systems

Praveen Ranjan Srivastava <sup>+</sup>

Birla Institute of Technology & Science, Pilani, India. 333031

(Received September 16, 2008, accepted October 30, 2008)

**Abstract.** Successful integration of classes results in correct functioning of object-oriented software. It can be possible that individual classes may function correctly but several new faults can emerge when these independent classes are integrated together. Typically, the complexity of an OO system lies in its object interactions, not within class methods which tend to be small and simple. As a result, complex behaviors are observed when related classes are integrated and several kinds of faults can arise during integration: interface faults, conflicting functions, and missing functions. Thus testing each class independently does not eliminate the need for integration testing. A large number of possible interactions between interacting classes may need to be tested to ensure the correct functionality of the system. We present a technique that enhances the integration testing of classes by accounting for all possible states of interacting classes. This is important as interactions may trigger correct behavior for certain states and not for others.

**Key words:** Software testing, UML, objects, class diagram, Collaboration Diagram

## 1. Introduction

The object oriented methodology for software development gives several benefits through its features namely, encapsulation, polymorphism and inheritance. Most of the rapid application development in today's world is done using tools which strictly follow object orientation. Though aforementioned pillars of object oriented development offers ease to the developers but it creates tough challenges for the testing teams. The reason being that individual classes may function correctly but several new faults can emerge when these independent classes are integrated together. UML has become an industry standard for object oriented analysis and design. It offers various diagrams that capture most relevant information of the system in context. These diagrams can be used individually or combined together to perform unit testing as well as integration testing. For example, the state chart diagrams can be used for unit testing of the classes [2] and interaction diagrams like collaboration and sequence diagrams can be used for integration testing of classes. Large software systems are often modular in nature. Though these modules can be individual classes or set of classes, the complete system functionality can only be achieved when these modules are integrated together. Typically, the complexity of an object oriented system lies in its object interactions, not within class methods which tend to be small and simple. As a result, complex behaviors are observed when related classes are integrated and several kinds of faults can arise during integration: interface faults, conflicting functions, and missing functions. Thus testing each class independently does not eliminate the need for integration testing. We propose a technique for integration testing of classes accommodating the state information associated with the classes which is otherwise ignored. For this purpose we will be combining UML interaction diagrams with state-charts. A Graph Based Test Model [GBTM] is devised by combining the two diagrams that captures both interaction between the classes and the states associated. Test paths are then generated for a particular scenario using a method which takes into account all transition coverage criteria. To prove the efficiency of the approach we compare the result achieved with the one which only takes interaction diagrams for the purpose of integration testing.

## 2. Different Levels of Testing

We discuss techniques and some of the research efforts that are supported at different levels of testing using UML diagrams. Emphasis is laid on integration level testing which also subsumes the approach of the

---

<sup>+</sup> E-mail address: [praveenrsrivastava@gmail.com](mailto:praveenrsrivastava@gmail.com), [praveenr@bits-pilani.ac.in](mailto:praveenr@bits-pilani.ac.in), [pra\\_ranjan@rediffmail.com](mailto:pra_ranjan@rediffmail.com)

paper.

**a) Unit Testing:** Instance of a class can be thought of as an individual unit of execution unlike procedural languages in which functions or procedures are the smallest possible unit of execution. Integration testing which is a higher level of testing activity can only proceed if the unit testing is performed thoroughly as unit testing lays the foundation by establishing minimal operability of units that will be integrated to produce system as a whole.

**b) Integration Testing:** As has been discussed earlier classes need to interact to build the whole system. This is where integration testing comes into picture. Integration testing is primarily concerned with testing the interactions between units. This essentially attempts to validate that classes, implemented and tested individually, provide the intended functionality when made to interact with each other. The concentration at this level is on interaction between classes through method calls. Unit testing does support but does not produce reliable results. It happens because some of the bugs remain dormant during unit testing and pop up when these units are made to interact to each other example missing functions, interface faults etc. And hence the need of integration testing is inevitable.

**c) Conformance Testing:** In order to fulfill specified requirements is known as conformance testing. Conformance testing defines correctness of the implementation with respect to the formal specifications.

### 3. The Proposed Approach

Different kinds of UML diagrams have been used for software testing from different perspective as was discussed in above section UML state chart have been widely used for testing the behavior of the individual classes. Similarly UML interaction diagrams have been used for integration testing. Testing through state chart only focuses on state based behavior and testing through interaction diagrams focus only on interaction among classes but they miss to uncover faults that arrive as a result of state dependent class interaction.

We propose an approach that uncovers such faults for example, inconsistent state of classes, the wrong initial state of a class and the wrong calling state of a class. Gallagher et al [3] perform state dependent interaction testing but their approach uses finite state machine and converts them to data flow graph. Traditional data flow analysis criteria are then applied to generate possible test paths. We will use UML interaction diagrams and state charts and built intermediate test model which is a graph and applied state transition coverage criteria to generate test path.

Well defined sequence passed among collaborating object is used to model the behavior of the system. In the context of UML this ordered sequence of messages is captured using interaction diagrams (collaboration and sequence). We will use these two diagrams interchangeably. States of the objects strongly influence their behavior when sending and receiving messages: Stack cannot be popped out if it is empty, similarly an element cannot be pushed if it is full.

Our technique will exercise all possible states of the object involved as many classes exhibit state dependent behavior. It consists of following activities:

- a) Generation of graph based test model – an intermediate model is generated from UML interaction diagram, and the corresponding state–charts.
- b) Test paths are then generated from the GBTM based on all transition coverage criteria.
- c) All selected test path are executed by using manually generated test data.

### 4. Specification of the GBTM

The GBTM is used to generate test specifications for class integration testing in the section below we describe how this test model is generated:

- a) Interaction diagrams contain messages and state chart contain the corresponding call event. State chart are in flattened form.
- b) GBTM contains multiple vertices where each vertex correspondence to an instance of a class in a distinct state (corresponding to state defined in state chart). This is essentially to capture the characteristics of model classes which receive a message in more than one state and exhibit distinct behavior for the same message in different states. On the other hand a non model class only requires a single vertex.
- c) There are two types of edges in GBTM test model: message and transition edges. Message edge

represent a call action between two objects and a transition edge represents a state transition of an object on receiving a message. Each message edge may lead to a state transition. The transition edges connect two vertices of the same class. Message edges include sequence number, associated operation, receiver object and the sender object. The transition edges include sequence number, associated operation, accepting state and sending state.

### 5. GBTM Construction

We present an example for the construction of GBTM. The system being discussed is an online shopping application. In this system, a customer can log in to the website to purchase books online. The customer is authenticated on the website. Authentication manager is the corresponding system module that takes care of this event. Once logged in the customer is assigned with a virtual session. The session is distinctly tracked using the session ID. The customer in his session acquires a shopping cart that he uses to add the books he wishes to purchase. Also the user can opt to remove the books he/she had added to cart previously. Once the customer is done with choosing of books, he/she then places the order for purchase of book. This transaction is done through credit card payment and on successful payment, transaction history log is generated. This completes one cycle of online shopping application.

The following is the class diagram of the system in context:

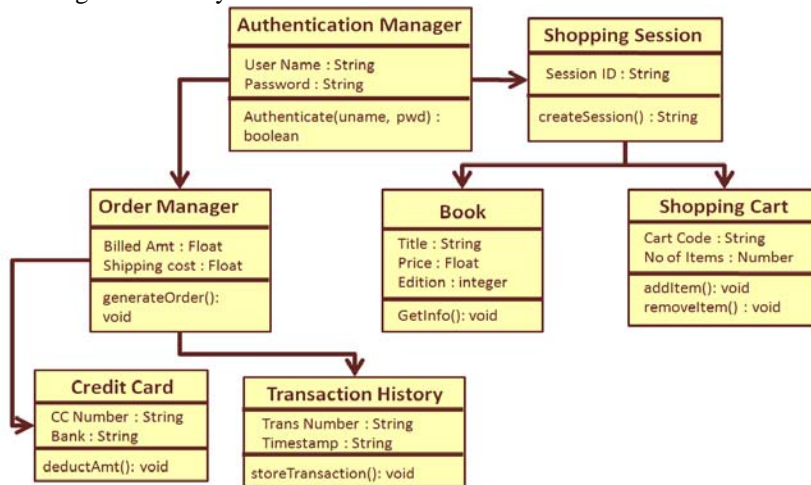


Figure1: Class Diagram of Online Shopping

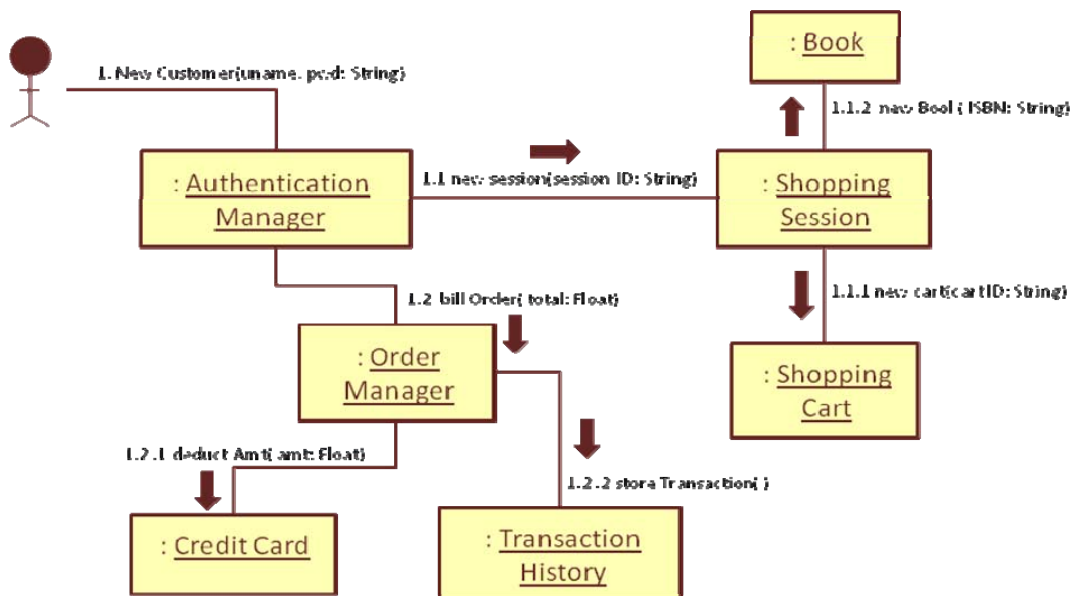


Figure2: Collaboration Diagram of Online Shopping

Figure 2 shows a collaboration diagram for the new Customer ( ) system level operation that instructs the online shopping system to award a fresh session to a customer who has arrived for shopping. An object of

the authentication manager class receives the operation call message with an argument specifying username and password. Several other classes also interact to provide the whole system functionality.

In the given collaboration diagram of figure 2 classes Shopping Session, Shopping Cart and Transaction History are modal classes. The rest of the classes are non-modal.

Collaboration diagram usually model the execution of a use case, triggered by one or more system level operations. The recipient of a system level message is generally a boundary object that forwards the message to one or more use case controller objects. Each message in a collaboration diagram is represented with a well defined sequence number, source, and destination objects. On the other hand, the statechart of an object defines its states and the messages it can receive in those states.

Figure 3 to 5 show flattened versions of the statecharts of the modal classes described above.

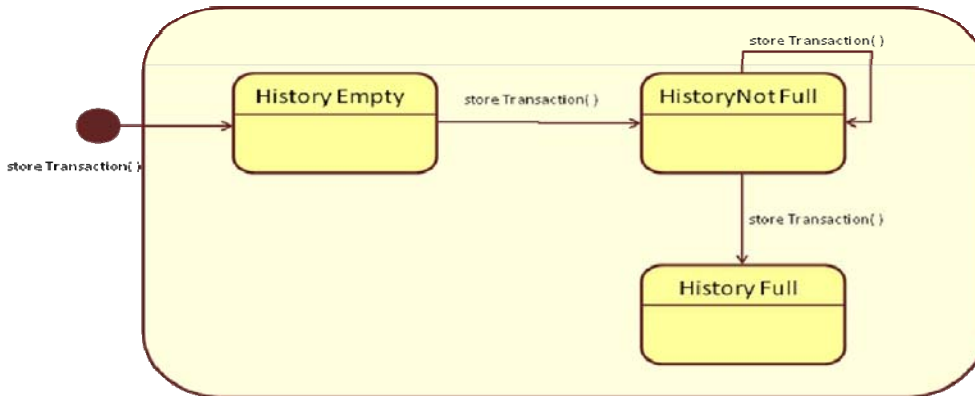


Figure 3: Statechart for Transaction History class

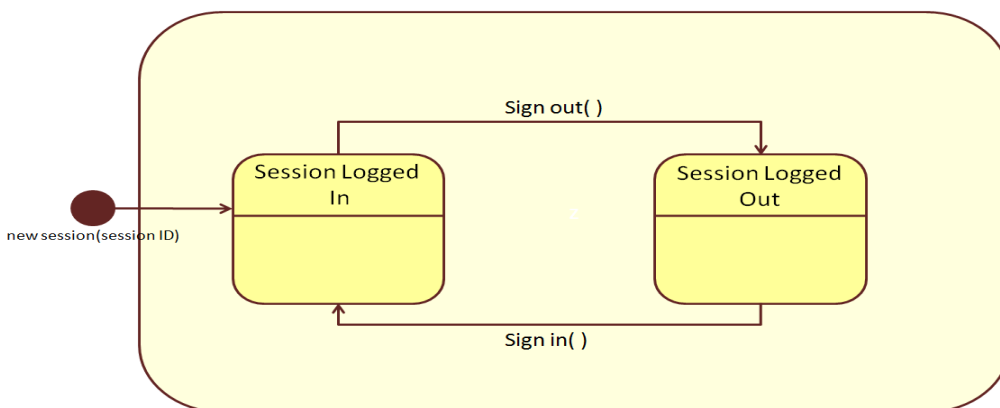


Figure 4: Statechart for Shopping Session class

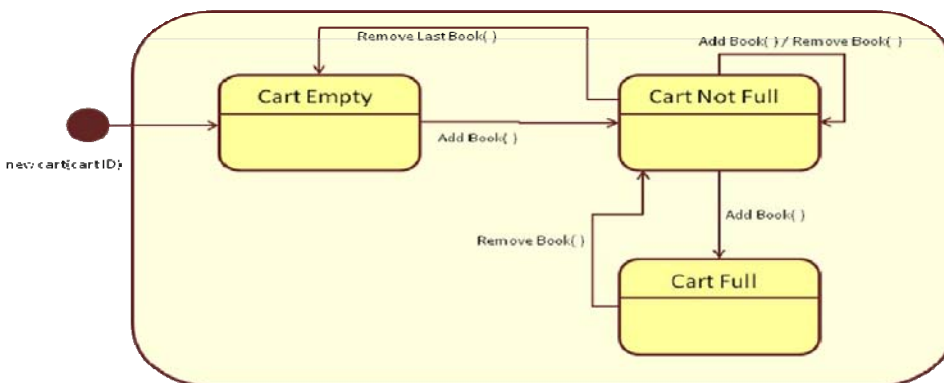


Figure 5: Statechart for ShoppingCart class

The GBTM outlines the chain of messages defined in the collaboration diagrams with the state

information of each participant object.

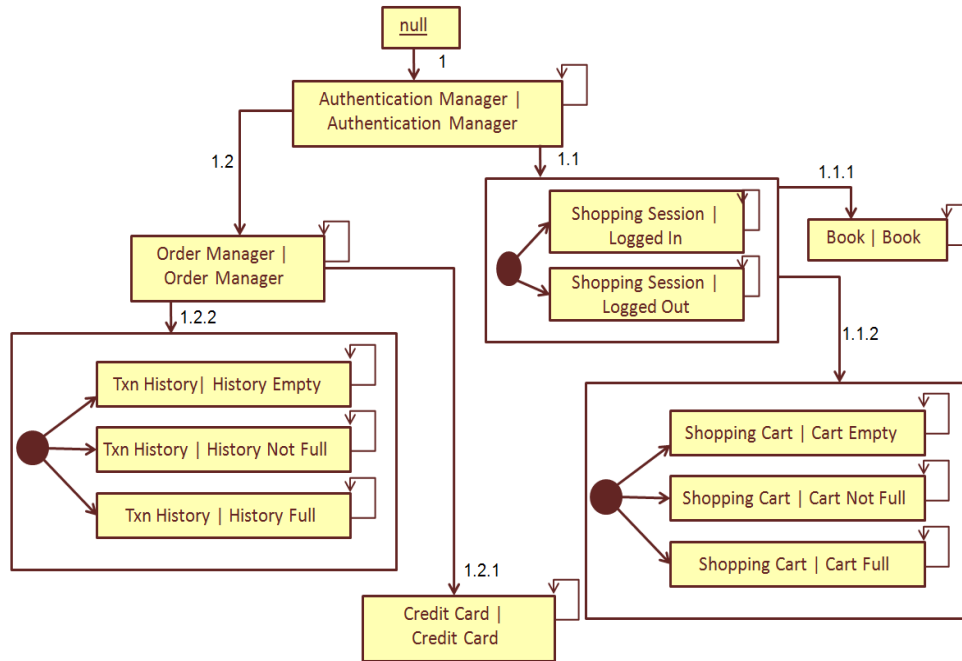


Figure 6: The GBTM for newCustomer( )

For constructing the GBTM for new customer( ) message, we start from the collaboration diagram of figure 2. Depending on whether the class is modal, one or more vertices in the GBTM are created. Multiple vertices for these modal classes in the collaboration represent various states in which the class can receive the incoming messages shown in the collaboration for that particular instance. For the purpose of simplicity the vertices corresponding to the same nodal class are grouped together within a box. A complete GBTM for a new customer( ) is shown in figure 6.

## 6. Test Path Generation from GBTM

This section will discuss test paths generation from the GBTM graph. Each path tests some interactions between classes in appropriate states. The traversal of GBTM tests all interactions between classes in all possible, valid states of classes involved in the particular collaboration. The total number of test paths can be calculated by taking the product of all transition paths of modal classes involved in the collaboration. In our case study of online shopping, we have three modal classes viz. ShoppingSession, ShoppingCart and TransactionHistory with 2, 3, 3 transitions respectively. So total number of test paths will be equal to eighteen.

## 7. Test Paths Coverage Criteria

We will take all transition coverage criteria into account. This criterion ensures that each state transition in a modal class is followed at least once. The next section presents the test paths generated by applying all transition coverage criteria on GBTM.

## 8. Comparing Test Paths Generated Using Collaboration Diagram Vs GBTM

Finally in this section we show the efficiency of GBTM approach against the traditional test path generation approach using only collaboration diagram. For this purpose we come out with all possible test paths for above case study using both approaches.

Test path generation using Only Collaboration Diagram

Test Path Sequence	No of collaborating classes	No. of test paths generated
1. newCustomer:AuthenticationManager 1.1. newSession:ShoppingSession 1.1.1 newBook:Book	3	3
1. newCustomer:AuthenticationManager 1.1. newSession:ShoppingSession 1.1.1 newBook:Book 1.1.2 newCart:ShoppingCart	4	4
1. newCustomer:AuthenticationManager 1.1. newSession:ShoppingSession 1.1.1 newBook:Book 1.1.2 newCart:ShoppingCart 1.2 billOrder:OrderManager 1.2.1 deductAmt:CreditCard	6	6
1. newCustomer:AuthenticationManager 1.1. newSession:ShoppingSession 1.1.1 newBook:Book 1.1.2 newCart:ShoppingCart 1.2 billOrder:OrderManager 1.2.1 deductAmt:CreditCard 1.2.2 storeTransaction:TransactionHistory	7	7

Figure 7: Collaboration specific test path generation

Test path generation using GBTM

Test Path Sequence	No of collaborating classes	No. of test paths generated
1. newCustomer:AuthenticationManager AuthenticationManager 1.1. newSession:ShoppingSession LoggedOut	2	2
1. newCustomer:AuthenticationManager AuthenticationManager 1.1. newSession:ShoppingSession LoggedIn 1.1.1 newBook:Book Book	3	3
1. newCustomer:AuthenticationManager AuthenticationManager 1.1. newSession:ShoppingSession LoggedIn 1.1.2 newCart:ShoppingCart CartEmpty	3	9
1. newCustomer:AuthenticationManager AuthenticationManager 1.1. newSession:ShoppingSession LoggedIn 1.1.2 newCart:ShoppingCart CartNotFull		
1. newCustomer:AuthenticationManager AuthenticationManager 1.1. newSession:ShoppingSession LoggedIn 1.1.2 newCart:ShoppingCart CartFull		
1. newCustomer:AuthenticationManager AuthenticationManager 1.1. newSession:ShoppingSession LoggedIn 1.1.1 newBook:Book Book 1.1.2 newCart:ShoppingCart CartEmpty	4	12
1. newCustomer:AuthenticationManager AuthenticationManager 1.1. newSession:ShoppingSession LoggedIn 1.1.1 newBook:Book Book 1.1.2 newCart:ShoppingCart CartNotFull		
1. newCustomer:AuthenticationManager AuthenticationManager 1.1. newSession:ShoppingSession LoggedIn 1.1.1 newBook:Book Book		

1.1.2 newCart:ShoppingCart CartFull		
<p>1. newCustomer:AuthenticationManager AuthenticationManager  1.1. newSession:ShoppingSession LoggedIn  1.1.1 newBook:Book Book  1.1.2 newCart:ShoppingCart CartFull  1.2 billOrder:OrderManager  OrderManager  1.2.1 deductAmt:CreditCard CreditCard</p> <p>1. newCustomer:AuthenticationManager AuthenticationManager  1.1. newSession:ShoppingSession LoggedIn  1.1.1 newBook:Book Book  1.1.2 newCart:ShoppingCart CartNotFull  1.2 billOrder:OrderManager  OrderManager  1.2.1 deductAmt:CreditCard CreditCard</p> <p>1. newCustomer:AuthenticationManager AuthenticationManager  1.1. newSession:ShoppingSession LoggedIn  1.1.1 newBook:Book Book  1.1.2 newCart:ShoppingCart CartEmpty  1.2 billOrder:OrderManager  OrderManager  1.2.1 deductAmt:CreditCard CreditCard</p>	6	18
<p>1. newCustomer:AuthenticationManager AuthenticationManager  1.1. newSession:ShoppingSession LoggedIn  1.1.1 newBook:Book Book  1.1.2 newCart:ShoppingCart CartFull  1.2 billOrder:OrderManager  OrderManager  1.2.1 deductAmt:CreditCard CreditCard  1.2.2 storeTransaction:TransactionHistory HistoryEmpty</p> <p>1. newCustomer:AuthenticationManager AuthenticationManager  1.1. newSession:ShoppingSession LoggedIn  1.1.1 newBook:Book Book  1.1.2 newCart:ShoppingCart CartFull  1.2 billOrder:OrderManager  OrderManager  1.2.1 deductAmt:CreditCard CreditCard  1.2.2 storeTransaction:TransactionHistory HistoryNotFull</p> <p>1. newCustomer:AuthenticationManager AuthenticationManager  1.1. newSession:ShoppingSession LoggedIn  1.1.1 newBook:Book Book  1.1.2 newCart:ShoppingCart CartFull  1.2 billOrder:OrderManager  OrderManager  1.2.1 deductAmt:CreditCard CreditCard  1.2.2 storeTransaction:TransactionHistory HistoryFull</p> <p>1. newCustomer:AuthenticationManager AuthenticationManager  1.1. newSession:ShoppingSession LoggedIn  1.1.1 newBook:Book Book  1.1.2 newCart:ShoppingCart CartNotFull  1.2 billOrder:OrderManager  OrderManager  1.2.1 deductAmt:CreditCard CreditCard  1.2.2 storeTransaction:TransactionHistory HistoryEmpty</p> <p>1. newCustomer:AuthenticationManager AuthenticationManager  1.1. newSession:ShoppingSession LoggedIn  1.1.1 newBook:Book Book  1.1.2 newCart:ShoppingCart CartNotFull  1.2 billOrder:OrderManager  OrderManager  1.2.1 deductAmt:CreditCard CreditCard  1.2.2 storeTransaction:TransactionHistory HistoryNotFull</p> <p>1. newCustomer:AuthenticationManager AuthenticationManager  1.1. newSession:ShoppingSession LoggedIn  1.1.1 newBook:Book Book  1.1.2 newCart:ShoppingCart CartNotFull</p>	7	63

1.2 billOrder:OrderManager  OrderManager 1.2.1 deductAmt:CreditCard CreditCard 1.2.2 storeTransaction:TransactionHistory HistoryFull  1. newCustomer:AuthenticationManager AuthenticationManager 1.1. newSession:ShoppingSession LoggedIn 1.1.1 newBook:Book Book 1.1.2 newCart:ShoppingCart CartEmpty 1.2 billOrder:OrderManager  OrderManager 1.2.1 deductAmt:CreditCard CreditCard 1.2.2 storeTransaction:TransactionHistory HistoryEmpty  1. newCustomer:AuthenticationManager AuthenticationManager 1.1. newSession:ShoppingSession LoggedIn 1.1.1 newBook:Book Book 1.1.2 newCart:ShoppingCart CartEmpty 1.2 billOrder:OrderManager  OrderManager 1.2.1 deductAmt:CreditCard CreditCard 1.2.2 storeTransaction:TransactionHistory HistoryNotFull  1. newCustomer:AuthenticationManager AuthenticationManager 1.1. newSession:ShoppingSession LoggedIn 1.1.1 newBook:Book Book 1.1.2 newCart:ShoppingCart CartEmpty 1.2 billOrder:OrderManager  OrderManager 1.2.1 deductAmt:CreditCard CreditCard 1.2.2 storeTransaction:TransactionHistory HistoryFull		
---	--	--

Figure 8: GBTM specific test path generation

Comparative Analysis of the Two Approaches

As can be seen from the graph, the exhaustiveness of GBTM approach is much higher than the conventional approach of using only collaboration diagram. With the GBTM approach more number of faults is likely to be seeded when the classes are made to collaborate to provide full system functionality.

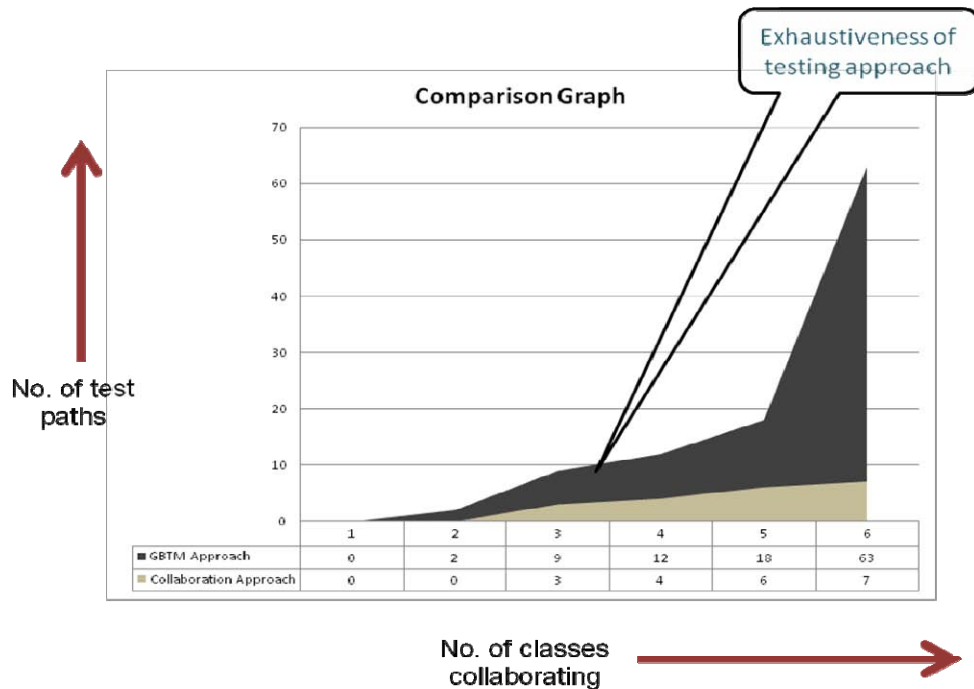


Figure 9: Comparative analysis of two approaches

9. Conclusion

In this paper we have combined UML interaction diagrams with state chart diagram to exercise class interactions in the context of multiple states of interacting classes in order to detect state faults which are

otherwise ignored.

## 10. References

- [1] [Kim99] Y.G. Kim et al. Test Cases Generation From UML State Diagrams. *IEEE Proceedings – Software*. 1999, **146**(4): 187-192,
- [2] [Pressman05] Roger Pressman. *Software Engineering: A Practitioner's Approach, 6th Ed.*. India: McGrawHill Higher Education, 2005.
- [3] [Gallagher06] A.J. Offutt, A. Cincotta. Integration testing of object oriented components using finite state machines. *Journal of Software Testing, Verification, and Reliability*, 2006
- [4] [Abdulrazik00] A. Abdulrazik, J. Offutt. Using UML Collaboration Diagrams for Static Checking and Test Generation. *The Third International Conference on The Unified Modeling Language (UML '00)*. 2000, pp. 383-395. York, UK.