Exploring the Need for Specialized Testing Technique for an Agent-Based Software

N.Sivakumar, K.Vivekanandan

Abstract*— The field of Software Engineering has been complimented with a number of research works that helps in developing software products that performs well within ever-changing organizational environments. Functional efficiency of a product greatly depends on the software development approach used to build it and the testing techniques involved. Most widely used software development approaches are conventional approach, object-oriented approach and agent-oriented approach. Among the Software Development Life Cycle (SDLC) phases, software testing is an important activity aimed at evaluating an attribute or capability of a program or system and determining that it meets the functionality of the system like the actual code or the non-functional requirements of the system like amicable user interface. There are various testing strategies used in the conventional approach like unit testing, integration testing, validation testing and system testing. As all the basic entities are viewed as objects and classes in the object-oriented software development, conventional testing approaches are not suitable and thereby specialized object-oriented software testing evolves. Recent literature study claims that none of the existing Agent Oriented Software Engineering (AOSE) methodologies possesses testing phase in their SDLC stating that the software developed using agent paradigm were been tested using either conventional or object-oriented testing mechanism. Since the agent characteristics such as autonomy, pro-activity, reactivity, social ability, intelligence etc., differs with object characteristics, object-oriented testing mechanisms are not sufficient and also inadequate to test the agent oriented software. Thus this paper compares various approaches of building and testing the software with the help of a case study (online book store application) and thereby the need for a specialized agent-oriented software testing mechanism is justified for the better functional outcome of the software product developed using agent oriented approach

***Index Terms*— Software Engineering, Agent-Oriented Software Testing, Software Development Life Cycle.**

I. INTRODUCTION

A software development methodology refers to the framework that is used to structure, plan, and control the process of developing a software system [1]. A wide variety of such frameworks have evolved over the years, each with its own recognized strength and weaknesses. Now an increasing number of problems in industrial, commercial, medical, networking and educational application domains are being solved by agent-based solutions [2]. The key abstraction in these solutions is the agent. An "agent" is an autonomous, flexible and social system that interacts with its environment in order to satisfy its design agenda. In some cases, two or more agents should interact with each other in a multi agent system (MAS) to solve a problem that they cannot handle alone.

Manuscript Received on November, 2012.

N.Sivakumar, Assistant Professor, Department of Computer Science and Engineering, Pondicherry Engineering College, Puducherry, India.

K.Vivekanandan, Professor, Department of Computer Science and Engineering, Pondicherry Engineering College, Puducherry, India.

According to the above mentioned facts, developing agent-based systems, developers will face new abstractions and concepts. Also they should handle main challenges that exist in the development of complex, open and distributed systems. For example, autonomy of agents, emergent behaviour and dynamic configuration are among the prominent features of multi-agent systems but in practice and in development time these features make system development more complex and practitioners need new methods and tools to handle these types of complexity.

Agent-oriented software engineering (AOSE) is a new discipline that encompasses necessary methods, techniques and tools for developing agent-based systems. It is a powerful way of approaching complex and large scale software engineering problems and developing agent-base systems. Several AOSE methodologies were proposed for developing software, equipped with distinct concepts and modelling tools, in which the key abstraction used in its concepts is that of an agent [3]. Few AOSE methodologies were listed below.

1. MAS CommonKADS (1996-1998)
2. MaSE(1999)
3. GAIA(2000)
4. MESSAGE(2001)
5. TROPOS(2002)
6. PROMETHEUS(2002)
7. ADLEFE(2002)
8. INGENIAS(2002)
9. PASSI(2002)
10. AOR Modeling(2003)

When we analysed and compared the strengths and weaknesses of the above mentioned AOSE methodologies, the strong weakness that we observed from almost all the methodologies were, there is no proper testing mechanism for testing the agent-oriented software [4]. Our survey states that the agent based software are currently been tested by using Object-Oriented (OO) testing techniques, upon mapping of Agent-Oriented (AO) abstractions into OO constructs. However agent properties such as Autonomy, Proactivity, and Reactivity etc., cannot be mapped into OO constructs. There arises the need for proper testing techniques for agent based software. The main objective of the paper is to analyse the existing testing approaches for testing an agent-based system and thereby the need for specialized testing technique is justified

II. OBJECT ORIENTED SOFTWARE TESTING

As an increment to the existing conventional approach, object-oriented approach was developed as the next great advance of software engineering. Software developed using object-oriented concept has a different structure and behaviour that the one developed using procedural languages.

Object-oriented approach is a data centric approach rather than algorithmic and it is a method based on hierarchy of classes and well-defined and cooperating objects [5]. An object can be defined as a thing or entity which can store data and send & receive messages whereas class can be defined as group of objects that share common properties and relationships. Everything in object-oriented programming is grouped as self sustainable objects, thereby gaining reusability by means of four important object-oriented features such as Encapsulation, Abstraction, Inheritance and Polymorphism. Encapsulation can be defined as wrapping up of data and functions into a single unit. Objects are described as implementations of abstract data types (ADTs). Usually an ADT definition is called class, while an object is a runtime instance of a class. Inheritance is the process by which objects of one class acquire the properties of another class. Polymorphism enables a number of different operations to have the same name i.e program entities should be permitted to refer to objects of more than one class, when a hierarchical relationship among these classes exits.

As in the conventional testing, unit, integration, validation and system testing are the testing levels involved in object oriented testing [6]. Though the levels are the same, the approach of testing widely differs. As everything in object-oriented software is viewed as objects and classes conventional testing cannot be accommodated in object-oriented software.

A. Object Oriented Unit Testing [7]

The primary aim of unit testing is to uncover errors within a given unit. In the context of object-orientation, the smallest unit may be a method or a class. Testing a method which is considered as a single operation of a class is termed as intra-method testing and testing the integrity of the class as a whole is termed as intra-class testing. A class is a combination of data members and member functions. Testing a unit may involve more than one class because a class can contain a number of different operations and a particular operation may exist as part of a number of different classes. Class testing is driven by operations encapsulated by the class and the state behavior of the class. Objects may also interact with one another with unforeseen combinations and invocations. These aspects of object-oriented features make the conventional testing unfit to test object-oriented software and thereby the need for object-oriented testing is justified.

B. Object Oriented Integrating Testing

Integrating the classes to be tested and verifying the class interaction, polymorphic calls and exception handling is termed as integration testing in object-oriented concept [8]. As class is considered to be a unit, integration testing is called as inter-class testing. As object-oriented software does not have any hierarchical control structure, the integration testing cannot be done either top-down or bottom-up as did in conventional integrated testing. Linear integration of classes cannot be performed in object-oriented software as there will be direct and indirect communication of components that make up the class.

There are two important strategies for integrated testing of object-oriented system such as Thread based testing and Use based testing. *Thread-based testing*, integrates the set of classes required to respond to one input or event for the system. Each thread is integrated and tested individually. Regression testing is applied to ensure that no side effects occur. *Use-based testing*, begins the construction of the system by testing those classes (called independent classes) that use very few of server classes. After the independent classes are tested, the dependent classes that use the independent classes are tested. This sequence of testing layers of dependent classes continues until the entire system is constructed.

C. Object Oriented Validation and System Testing

There is no significant difference between the conventional software testing and object-oriented software testing with respect to validation level and system level testing since both (conventional & object-oriented) the tests focuses on user-visible actions and user-recognizable output from the system. Conventional black box testing can be applied for validating the object-oriented software.

III. AGENT ORIENTED SOFTWARE TESTING

Agent-oriented paradigm can be considered as the extension of object-oriented paradigm [9]. There is a growing need for agent-oriented system to tackle complex problems. A software agent is defined as a software program that can perform specific tasks for a user and possessing a degree of intelligence that permits it to perform parts of its tasks autonomously and to interact with its environment in a useful manner. More than one agent is composed together and interaction is exhibited among themselves to achieve the targeted goal is termed as a Multi-Agent System (MAS). MASs are meant for building complex distributed system. An agent can be termed as an intelligent object due to the fact that agent possesses certain unique **a**nthropomorphic characteristics such as autonomy, pro-activity, reactivity, social ability, mobility, etc.,

Autonomy: The ability to take goal directed autonomous decisions without intervening with human or other agents.

Pro-activity: The ability to initiate actions to accomplish the goals rather than responding to the unpredictable environment.

Reactivity: The ability to respond in selective and timely fashion by perceiving the environment.

Social ability: The ability to work in collaboration with other agents and human in order to achieve the goal.

Mobility: The ability to move from one platform/environment to another in self-directed way.

Testing an agent-oriented system is a complex task and very little work is been carried out so far [10]. There are plenty of agent oriented software engineering methodologies that deals about how an agent based system is analysed, designed and implemented. There is no testing framework explored in any of the existing methodologies stating that the agent oriented software can be tested using the existing object-oriented testing techniques. Though objects and agents looks similar they differ widely too.

A. Agent Oriented Unit Testing

In a multi-agent system it may not always be possible to scan/gather the complete agent requirements, due to the fact that agent changes its functionalities based on the environmental need so as to achieve the goal.

Agent is an important building block of a multi-agent system and it is very essential to verify whether the agent in isolation matches with the specification under normal and abnormal condition. Thus in an agent-oriented programming, the smallest building block/unit is an agent itself. Unit testing with respect to agent based system is to individually test all the agents involved in the system. Testing an agent is based on its own mental attributes such as goal, plan, role and action. Test cases are to be generated so as to ensure the following,

1. To test whether the goal of an agent is achieved.
2. To test whether the plan get triggered by the event that is supposed to handle.
3. To test whether the agent takes appropriate role as expected by the environment
4. To test whether the agent performs the activity as per the functionality.

B. Agent Oriented Integrated Testing

Multi-agent system is a logical collection of agents that interact with each other in a way that implements the functionality of the system. After ensuring that the individual agent in isolation is working as per the requirement, the next immediate step is to integrate the agents involved in the MAS so as to test the interaction, and communication among agents. The protocol involved in communication among agent is also tested during integration.

A. Agent Oriented System Testing

System testing in agent oriented approach will test the complete functionality and test the system as a whole. Here the perceptions and actions of all the agents are tested as a whole by providing proper test cases.

IV. CONVENTIONAL VS OBJECT-ORIENTED TESTING

Based on the survey from various literatures comparison is made between the approaches of conventional and object-oriented testing and we found more reasons why testing approach followed in conventional software cannot be applied to object-oriented software. Difference between conventional testing and object-oriented testing are tabulated below.

Table I. Difference between conventional and object-oriented software testing

Sl. No	Conventional Testing	Object-Oriented Testing
1.	Module / Subroutine / are considered as unit	Class is an unit
2.	Single operation of a module in isolation can be tested	We can no longer test a single operation in isolation but rather as part of a class
3.	Has hierarchical control structure	Does not have hierarchical control structure
4.	Top-down or bottom-up integration is possible	Ordering cannot be followed
5.	Incremental integration is possible	Incremental integration is not possible

V. OBJECT-ORIENTED VS AGENT-ORIENTED TESTING

Though it is considered that agent-oriented paradigm is a natural extension of object-oriented paradigm, the properties of an objects and agents differ widely [11]. Agent is termed as an intelligent object since agent possesses few unique properties such as pro-activity, reactivity, autonomous, social ability, learnability etc. This intelligence factor of agent makes testing more complex and thus it is difficult to incorporate object-oriented testing techniques for agent-based software and there arises the need for specialized testing techniques. The kind of interaction between objects and between agents is represented in Fig.2. The following table explores the major differences of an object and agent.

Table II. Difference between Object-oriented and Agent-oriented software testing

Sl. No	Object Oriented Software	Agent Oriented Software
1	Basic unit is object	Basic unit is agents
2	Defined by methods and functions	Defined by their behavior
3	Extension of structured programming. Persist data hiding, class hierarchy and data encapsulation	Extension of object oriented programming. Have independent thread of control and initiative
4	In OO languages, objects are created by a class. Once created, may never change their class or become instances of multiple classes. Thus static.	Agents are been created to achieve a certain goal. The agent may change its behaviour according to the situation. Thus it is dynamic
5	While object systems are be built to include these particularly the IF-THEN rules of expert systems	Agent-based systems can support concepts such as rules, constraints, goals, beliefs, desires, and responsibilities
6	Less resistance to failure	Robust and finds a way to come out of failure
7	Relations are collaboration, composition, inheritance, instantiation and polymorphism	Relation between agents is negotiation, role multiplicity, individual knowledge, service match making.
8	Lack perseverance. Does the work assigned by user.	Preserving in nature. Understand the content of the problem and reacts accordingly.

VI. CASE STUDY

The main objective of the paper is to justify the need for specialized testing technique for agent oriented system. To explore the need for agent oriented software testing, an online book store application has been considered as a case study.

The scope of the application is to develop a fully online system for worldwide sale of books, which enables an end user to browse for books online, add books to a shopping cart, and place an order for the books in the cart. This system will offer a broad range of books to customers, and a personalized friendly user interface. The system facilitates fast and reliable service at all stages, from locating a desired book, to deliver of the purchase.

A. Conventional Development

In traditional / conventional approach, the book store application is built using various software development phases like requirement analysis, design, implementation and testing. The requirement is usually done by identifying the stakeholders, recognizing the multiple view points, working towards collaboration and asking the preliminary questions. The requirements analysis of the book store include identifying the users, the type of books available, the interface, catalogue maintenance, update of the books, stock maintenance, customer details, details of the author and publisher, placement of orders, delivery of goods, receipt maintenance, payment modes, etc.

Design engineering encompasses the set of principles, concepts that lead to the development of a high quality software product. A prototype of the product is developed in the design phase, for the user to understand how the culmination of the product will appear. Design prototype of the book store is show in Fig.1.

Implementation phase comprises the actual coding which implements the functionalities specified. A structured programming language is employed for this purpose.

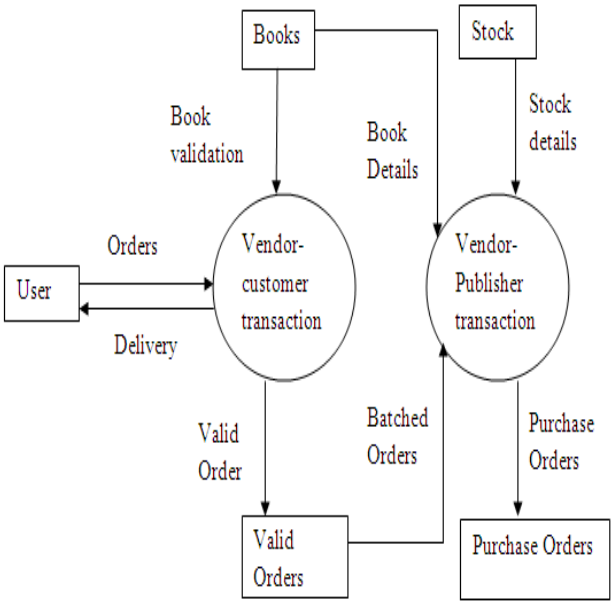

Fig 1. Process Model for online book store

B. Conventional Testing

The conventional testing for the book store application is done as follows: The unit testing begins by splitting the book store application into various modules like order placing, delivery, stock maintenance, payment modes, etc. The functionalities of these modules are tested in isolation. Every module is taken one at a time and tested individually with an aim to locate bugs. If any discrepancy occurs, the errors are rectified and the module is tested again. Testing a module could be iterative and may have no constraints on the number of times it is tested. Once the module has been tested and believed to be error free, the next module is taken up for testing. This process continues all until all the modules in the book store application are tested individually.

The integration testing begins by grouping the modules like the order, stock and its interface and tested. It should be noted that these modules have already been tested individually and believed to be error free. When these separate modules are integrated, we test the communication, data flow and the interface between these modules and observe the way they work as a whole. If any error occurs in the communication channel or in the grouping, then those specific errors are located and rectified. After this testing is successful, the delivery module is added in an incremental fashion and the new integrated system is now checked on the whole for functional specifications. The errors, if any, are rectified and the process continues until all the modules in the system are unified and tested.

System testing follows integration testing. The system is tested as an agglomeration of all the modules present in the design along with their functionalities. At every stage, bugs are located and corresponding corrections are made. The modules of the book store like placing order, stock delivery, books, catalogue, etc. developed in the implementation phase are unit tested and integrated to undergo integration testing. The error free modules are installed and the system as a whole is tested. It also includes recovery testing, security testing, stress testing and performance testing. Stress testing is when a large number of orders are placed, the response pattern of the 'placing order', 'purchase' and 'delivery' modules are tested and observed. The time interval between occurrence of a fault and its recovery forms the main criteria of recovery testing. Suppose the stock counts go negative, a system error or fault occurs. The time taken for the stock to gain a positive number and time taken by the system to recover is termed as recovery testing.

In the acceptance testing, before the product release, it will be tested by placing an order, checking the stocks and delivery of the books at different scenarios. If the user is not satisfied with the behavior of the system or the modules involved, then some modifications to the existing design is necessary.

The black box testing is done by a series of inputs applied to the book store system (placing an order) and corresponding output is noted and verified (delivery of books if the stock is available). The inputs are given and the outputs generated help to locate the possibility of errors in the system.

The white box testing is performed by giving inputs like making purchase orders for books from the publisher, that fall below the minimal level of stock. The logical correctness of the module is verified by noting the output.

C. Object-Oriented Development

The intent of object oriented analysis (OOA) is to define all classes (and the relationship and behavior associated with them) that are relevant to the online book store application. To accomplish this, a number of tasks are put forth:

1. Basic user requirements must be communicated between the customer and the software engineer.

2. Classes must be identified (i.e. methods and attributes are identified).
3. A class hierarchy is defined.
4. Object-object relationship (object connections) should be represented.
5. Object behavior must be modeled.
6. Tasks 1 through 5 are re applied iteratively until the model is complete.

The OO analysis includes writing use cases, developing activity diagram, state transition diagram, class diagram and other UML diagram. A scenario is a very specific sequence of actions and responses that detail how the system will interact with a user. It is useful for pinpointing vague or missing requirements, and for verifying that the system does what's expected of it. A complete system would have several top-level scenarios, each showing a typical interaction with the system. The scenarios are the basis for detailed requirements.

The scenario for the online book store application is:

- User searches for a title by browsing or keyword search
- System displays information about the title.
- User selects a title to buy
- User places the order
- System displays shipping address and billing address
- User confirms order and payment method.
- System processes order, notifies warehouse for shipping, and issues an electronic receipt.

The analysis model defines a complete set of classes that describe some element of the problem domain. As the design model evolves, it is necessary to define a set of design classes that i.) Refine the analysis class by providing design detail that will enable the classes to be implemented, ii.) Create a new set of design classes that implement a software infrastructure to support the business solution. The different design classes are;

1. User interface class
2. Business domain class
3. Process classes
4. Persistent classes
5. System classes

The classes formed in the analysis phase are refined to avoid redundancy and reduced as much as possible to classes in the design phase. The class relation diagram includes all classes and relations between the classes. The class relation diagram for an online book store is shown in fig 2.

The final product of the design phase is the general architecture of the application. Although there are many user interface design models, there are certain steps to be followed like:

1. Using information developed during interface analysis, define interface objects and actions.
2. Define events (user actions) that will cause the state of the user interface to change the model of this behavior.
3. Depict each interface state as it will actually look to the end user.
4. Indicate how the user interprets the state of the system from information provided through the interface.

The code for the system is usually generated based on the architecture of the design. The code may be in any of the high level object oriented language like C++ or Java.

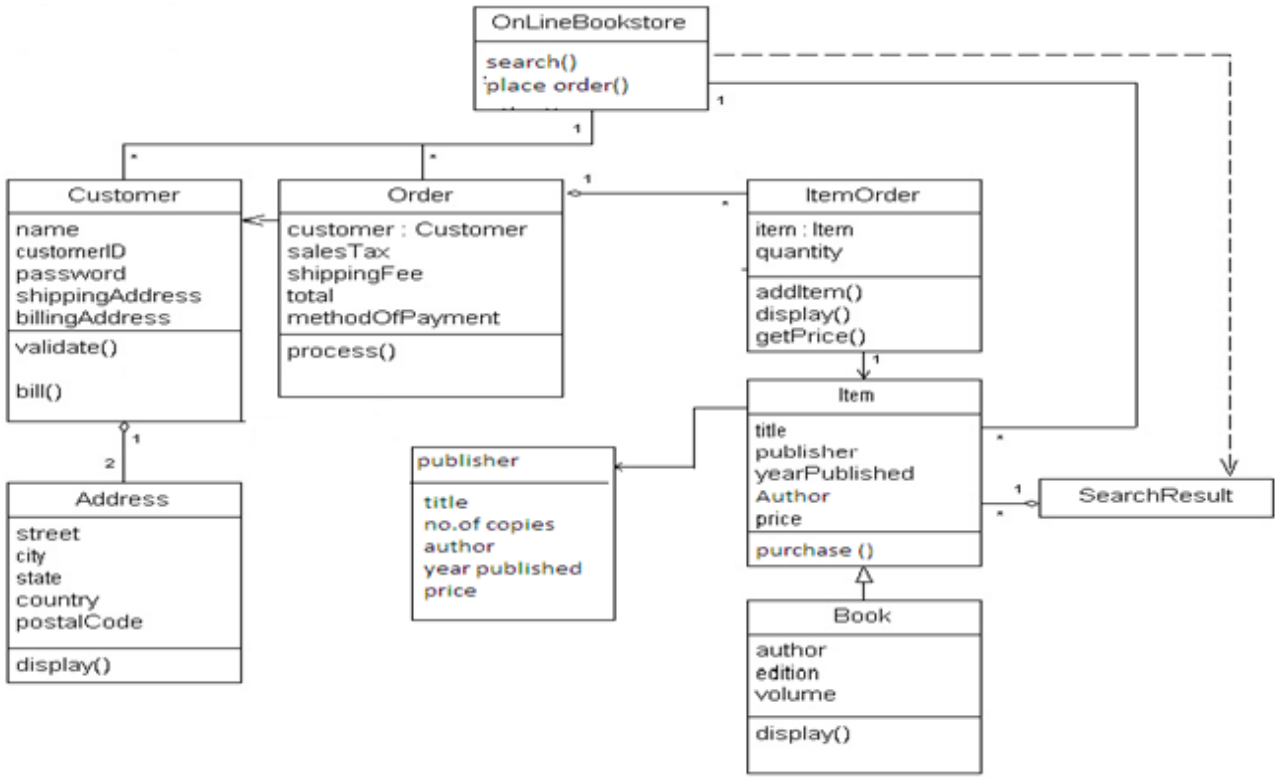

Fig 2.Class Interaction diagram for online book store

D. Object-Oriented Testing

The object oriented testing process begins with syntactic semantic and consistency testing. For the given book store application, we verify the semantic correctness by giving inputs like various names of book both which are valid and invalid. The syntactic correctness is verified by examining the syntax of the code. Consistency is evaluated by examining whether the customer receives the ordered book.

The integration testing, for book store application can be performed by testing events like purchase order ( ). It checks for the stock level and places order to the publisher accordingly, by invoking the necessary threads. Use based testing is achieved by testing independent class like the address class followed by testing its dependent class; the customer class. Validation testing for book store can be performed if the final system satisfies the user requirements.

System testing for book store is complimented by posting queries with large processing time and by simultaneously issuing enormous queries to the system. Based on the response and performance of the system to these queries and time taken to process these queries, stress and performance parameters are evaluated. This is almost similar to the testing done in traditional system except that the objects are tested finely for their properties and behavior in OO testing methodology. Security testing is very important in applications like book store where cash transactions are involved. It is necessary to maintain secure information about the transaction time, date, the amount involved and above all, the details of the parties involved in it. When a customer purchases a book, it is necessary to verify his/her identity but it is equally important to safeguard and ensure security of the user's identity.

E. Agent-Oriented Development

The bookstore application has been developed in agent oriented approach using Prometheus methodology [13]. Goals are central to the functioning of software agents that realize the system. The use of goals in requirements engineering or system specification phase facilitates a mapping into the detailed design and implementation, as well as providing mechanism for requirement specification. In addition it also requires specifications of functionalities related to the identified goals.

The initial set of goals identified for online book store application is

- Fully online system
- Wide range of books
- User friendly interface
- Purchase of books
- Delivery of books
- Competitive prices

Goals are refined iteratively to be more specific. Often similar sub goals arise under different initial goals. A grouping of similar sub goals provides the basis for "functionalities". For example, refining the original goal of fully online system, we obtain sub goals like pay online, order online, find books, while the original goal of purchase books leads to find books, place order, make payment and arrange delivery. Pay online and make payment are closely related and are therefore grouped together.

We continue to work with the list of goals and sub goals, coalescing similar goals and adding goals as we see that a particular grouping is lacking some aspect. For example, the sub goals of deliver internationally, courier delivery and arrange delivery are coalesced into single sub goal 'arrange delivery'.

Functionality includes grouping of related goals, percepts, actions and data relevant to the behavior. Functionalities allow for a mixture of both top-down and bottom-up design. They are identified by top-down process of goal development. At the same time, they provide a bottom-up mechanism for determining agent type and their responsibilities. The process of designing and grouping goals sets the initial stage of functionalities. Once the functionalities have been identified, functionality descriptors are developed.

Scenarios are complementary to goals. Scenario development involves how the goals are a part of various processes within the system. Scenarios help to identify the missing goals. Scenarios are used primarily to illustrate the normal running of the system, although it can be useful to develop some scenarios that indicate what is expected to happen when something goes wrong. As scenarios are developed, it is common to identify additional goals that are needed. Examples of scenarios developed for our system are; stock order scenario, book finding scenario, new catalogue scenario, order book scenario, etc. The architectural design provides the basis for high level design. There are 3 aspects developed during architectural design

1. Deciding agent types
2. Describing the interactions between the agents
3. Designing the overall system structure

Some interactions between the agents may be based on message passing, possibly for transfer of control as well as data. In our application, sales transaction functionality may pass control to delivery management functionality once it has completed its job. The architectural design defines what agents are to be a part of the system and method of interaction between these agents to meet the required functionality of the system. A major decision that is made during the architectural design is the types of agents used. Agent types are formed by combining functionalities. The choice of how functionalities are to be combined is made by considering functionalities and scenarios and developing possible groupings of functionalities into agents, which are then evaluated according to the standard criteria of coupling and cohesion. Coupling is the property of a group of components.

In agent systems, coupling is mainly exhibited in communication between agents. On the other hand, cohesion is a property of a single component. A component is cohesive if all of its parts are related. In finalizing the architectural design, deciding the interfaces play a major role. Interface description includes perceptions and actions. Percepts often require some processing in order to extract the information that is of value to the agent system. In systems where percepts originate from physical sensing devices of some sort, the data often contains the significant amounts of noise, which may require the use of techniques to filter and cleanse the data. Actions may also be complex, requiring significant design and development outside the realm of the reasoning system. Typically a system will have a number of perceptions and actions that forms the initial definition of how the system will interact with the environment and it is extremely important to ensure that these interfaces are correct and or achievable.

Following are the percepts and actions identified for the electronic bookstore. The percepts are bank transaction response, cheaper price report, stock arrival, new catalogue, etc. the various actions involved are bank transaction, deliver book, request delivery tracking, place delivery request, email stock order, etc. In the electronic bookstore, two external data are identified and they are Courier DB and Postal DB. In addition the other clusters of information are Customer DB, Customer orders, Pending orders, Delivery problems, Books DB, Stock DB, Stock orders. The various agents are identified in this phase and are used for online book store application. They are basically customer relations agent, delivery manager agent, sales assistant agent and stock manager agent. Each agent has its own functionalities, actions and protocols through which it communicates with the other agent. For example, sales assistant agent has its functions like bank transaction and page display for the customer. For this it uses customer database. This agent has to percept user input, arrival at their site and bank transaction response. Book finding, book ordering, update customer profile are the various protocols it uses to communicate with stock manager, delivery manager and customer relations agent respectively. It also uses various other protocols for different scenario. The initial data coupling diagram is shown in fig.3.

In the detailed design capabilities, plans and events of the agents are analyzed. The main steps in the final stage are

1. Decomposition using capability overview diagram
2. Sub tasks and alternative plans
3. Developing events and messages
4. Details of actions and percepts
5. Details of data

Stock management capability is broken down into ordering, delay handling and handling new stock. When a book is ordered it is important for that event to carry information as to which book is being ordered.

However not all of these design entities are carried to implementation. For example, functionalities are used to determine the agent types but they do not correspond to any run time entity. Also messages, plans and beliefs do not directly map into object oriented languages. Although it is possible to implement an object oriented design using an OO language, it is difficult to realize and maintain. Hence we make use of a range of agent oriented programming language [12] like JACK, JADE, etc.

F. Agent-Oriented Testing

Although the agent technology is gaining popularity there is a lack of proper testing mechanism for agent based systems. Unit testing will test the functionality of a particular agent. In the case of delivery manager agent, its functionalities like placing the delivery request and requesting for delivery tracking is tested and verified.

Consider the integration testing of modules like sales assistant agent and stock manager agent. They need to communicate via the protocol 'book finding'. The integration test will identify the bugs; if there is any problem in communication between the agents.

In the example considered, the system testing will test the percepts like failed stock arrival, cheaper price report, new catalogue, stock arrival, and regular order trigger. In a similar way, its actions like email of the stock order are also tested.

Though the testing strategies such as unit, integration and system testing are the same for conventional, object-oriented and agent-oriented software, the testing approach might differ widely. Presently, an agent system is usually tested using the traditional and object oriented testing strategy. Though objects and agents share similar characteristics, they also differ widely. Agent is termed as an intelligent object as agent possesses certain unique properties such as intelligence, autonomy, pro-activity, reactivity, mobility etc., Currently agent oriented software were tested using object-oriented testing technique upon mapping Object-Oriented (OO) constructs into Agent-Oriented (AO) constructs. However agent properties such as Autonomy, Pro-activity, and Reactivity etc., cannot be mapped into OO constructs. Moreover at certain instance, an agent responds differently for same input at different scenario. These situations cannot be handled by object-oriented testing techniques.

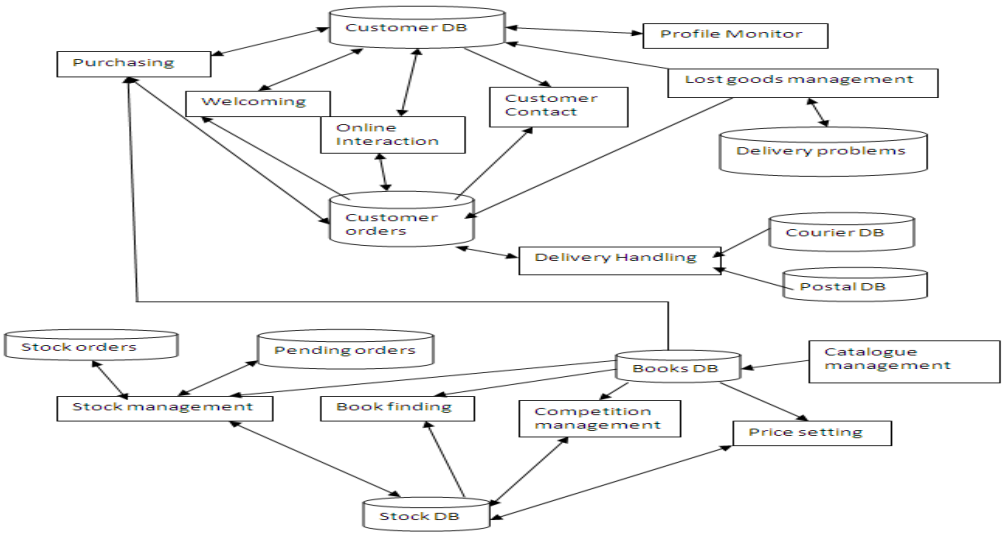

Fig.3 Data Coupling diagram for online bookstore

VII. CASE STUDY INTERPRETATION

Testing is an important activity in software development life cycle. However, testing is often disregarded in agent oriented atmosphere and the main focus is shifted to design and analysis activities alone. As per existing scenario, to test an agent oriented system we employ the existing testing techniques like the traditional testing and the object oriented testing. There is an allusion that the testing methods of the traditional approach can be made use of for testing the software developed in agent oriented approach as well. There is no specific testing technique developed for agent oriented approach. The existing testing strategies may help to test the functionality of the system but fail to efficiently test the properties of agents. They lack efficacy in determining the agent interaction and autonomous functionality of the agent. The various agent properties like heterogeneity, dynamism, autonomy, proactive and reactive behavior etc, cannot be tested glibly and with ease by the existing testing strategies. These properties of agents are the most important and highlight of the agent oriented approach. If these properties are not tested in detail, then the overall performance of the agent system is pulled down by the possible errors that are left undetected or unexamined. To reach out to all the possible errors and rectify them; is facilitated only by a testing strategy designed especially for agent oriented approach.

Table III. Life Cycle coverage of various development paradigm.

	Traditional	Object oriented	Agent oriented
Analysis	✓	✓	✓
Design	✓	✓	✓
Implementation	✓	✓	✓
Testing	✓	✓	**?**

VIII. CONCLUSION AND FUTURE WORK

The book store application under consideration has aided effectively in pin pointing the advantage and disadvantages of various approaches. The traditional model has been accompanied by its own requirement analysis, design, development, implementation and testing phases. The paradigm shift to object oriented approach has been complimented with its requirement analysis, design, development, implementation and testing methods. Unfortunately, the agent oriented approach has not been able to achieve the plethora of its development cycle. A separate agent oriented testing strategy is the need of the hour. This would complement the agent oriented development phases and improve the overall efficiency of agent oriented software in building complex systems effectively and with ease.

REFERENCES

1. Roger S.Pressman, "Software Engineering – A Practitioner's Approach", Tata Mc Graw Hill, 6th edition, 2005.
2. Nicholas R.Jennings, Michael Woolridge, "Agent oriented software engineering",Queen Mary and Westfield college, University of London.
3. Federico Bergenti, Marie Pierre Gleizes, Franco Zambonelli, "Methodologies and Software Engineering for Agent Systems-book", Kluwer Academic Publishers.
4. Chia-En Lin, Krishna M.Kavi, Frederick T. Sheldon and Thomas E. Potok, "A Methodology to Evaluate Agent Oriented Software Engineering Techniques", Computer Science and Engineering department, University of North Texas.
5. Bray, Mike. "Object-Oriented Design", Carnegie Mellon Software Engineering Institute, Feb 2008.
6. James.A.Whittaker, "What is software testing? And why is it so hard?", Florida Instittute of Technology,IEEE 2000.
7. Shalini Gambhir, " Testing strategies for Object-Oriented Systems" , International Journal of computer Science and Information Technology & Security, Vol.2, No.2, April 2012.
8. Zhe (Jessie) Li and Tom Maibaum, "An Approach to Integration Testing of Object-Oriented Programs", Department of Computing and Software McMaster University, Hamilton, ON, Canada, IEEE 2007.
9. Praveen Ranjan Srivastava, Karthik Anand V, Mayuri Rastogi, Vikrant Yadav, G Raghurama, "Extension of object oriented software testing techniques to agent oriented software testing", Journal Of Object Technology, Birla Institute of technology and science, Pilani, India.
10. Mailyn Moreno, Juan Pavon, Alejandro Rosete. Testing in Agent Oriented Methodologies. IWANN 2009, Part II, LNCS 5518, pp. 138–145, 2009.© Springer-Verlag Berlin Heidelberg 2009
11. Levine, David, "Relationship between Agent and Object Technologies", Agent technology green paper, OMG Agent work group, 1999.
12. Shoham, Y. Agent oriented programming (Technical Report STAN-CS-90-1335) Stanford University: Computer science department, 1994.
13. L. Padgham, J. Thangarajah, and M. Winikoff: "The Prometheus Design Tool – A Conference Management System Case Study." In: M. Luck and L. Padgham (Eds.): AOSE 2007, LNCS 4951, pp. 197–211, Springer-Verlag Berlin Heidelberg 2008.