Comparing the Testing Approaches of Traditional, Object-Oriented and Agent-Oriented Software System

N.Sivakumar[1] and K.Vivekanandan[2]
Department of computer Science and Engineering
Pondicherry Engineering College
Puducherry, INDIA
sivakumar11@pec.edu[1], k.vivekanandan@pec.edu[2]

Abstract—**Testing is one of the important and indispensable activities of software development life cycle. Software testing makes sure that the developed software satisfies all the customer requirements and executes without error. More researches in the field of software testing are carried out in academic as well as in industry so as to improve the testing process. As there is a major shift in the software development approach from conventional to object-oriented and to agent-oriented, testing has also shifted from conventional testing to object-oriented testing and to agent-oriented testing. Though the objective of the testing is the same for all different software development paradigms, the approach of testing may differ widely. The main objective of this paper is to analyze and compare the various approaches of testing process carried out in conventional software development, object-oriented software development and agent-oriented software development.**

Keywords- Software Testing, Software Agents, Object-Oriented System, Agent-Oriented System.

I. INTRODUCTION

The main goal of software testing is to find errors in a systematic and effective way rather that random experimentation [1]. Testing can be stated as the process of validating and verifying the software product whether it meets the business and technical requirements that guided its design and development, so that it works as expected. Verification is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase whereas validation is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements. The quality assurance of the software is much more dependent on the testing process. Software testing is adopted to uncover as many errors as it can and also to ensure the correctness of the software. The major activities involved in software testing are 1) Deriving test cases, 2) Executing the program using the generated test cases, and 3) Evaluating the test result. Test cases are the basis for testing any software but the approach of deriving the test case may vary upon different software development paradigm such as conventional, object-oriented and agent-oriented.

A. Testing Levels

Irrespective of using any software development paradigm, testing is performed at four important levels namely unit testing, integration testing, validation testing and system testing [2] as represented in Fig. 1.

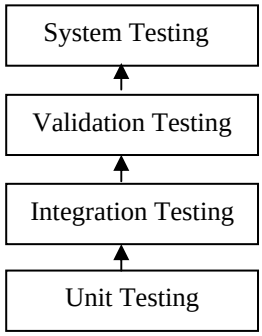

Fig. 1 Testing Levels

A unit is a smallest testable part of an application. In conventional/procedural programming, a unit may be an individual program, module, function, procedure, etc., while in object-oriented programming; the smallest

unit is a encapsulated class, which may belong to a base/super class, abstract class or derived/child class. In an agent-oriented programming, the smallest building block is an agent itself. Testing an agent is to verify whether the agent in isolation matches with the specification under normal and abnormal condition.

After the units are individually tested successfully, the next level of testing i.e integration testing proceeds. Integration testing in conventional software development is that to progressively integrate the tested units (module, program, procedure, function) either incrementally or non-incrementally, so as to check whether the software units in an integrated mode are working properly. In an object-oriented software development, integrated testing is to verify the interaction among classes (interclass). The relationships among classes are the basic characteristics of an object-oriented system and define the nature of interaction among classes and objects at runtime. When it comes to agent-oriented software, agents involved in a multi-agent system are integrated so as to test the interaction, and communication among agents.

Validation testing is followed by integration testing. Validation testing focuses on user-visible actions and user-recognizable output from the system. It demonstrates conformity with requirements. Black-box testing is a testing type used for validation purpose. Since black-box testing is used for validating the functionality of software, it is otherwise called as functional testing. Equivalence partitioning and Boundary value analysis are the two broad categories of black-box testing. There is no distinction among conventional, object-oriented and agent-oriented software with respect to validation testing

System testing verifies that all elements (hardware, people, databases) are integrated properly so as to ensure whether the overall product met its requirement and achieved the expected performance. System testing also deals with non-functional requirements of the software such as recovery testing, security testing, stress testing and performance testing.

B. Testing Types

There are two basic types of software testing namely black-box and white-box testing. Black box testing (also called functional testing) focuses solely on the outputs generated in response to selected inputs and execution conditions whereas white box testing (also called structural testing and glass box testing) is testing that takes into account the internal mechanism of a system or component. The test case design for black-box testing concerns itself exclusively with the inputs and outputs of an application thereby focusing only on the interface and functionality of the software rather than the internal working of the software. The test case design for white-box testing concerns itself with the internal working of the software i.e (1) to ensure whether all the independent paths exercised within a module or unit is traceable; (2) to ensure that all the logical decisions (true and false) involves proper branching; (3) execute loops at their boundaries and within their operational bounds; and (4) exercise internal data structures to ensure their validity [2]. Black-box and white-box testing is applicable at unit and integration level whereas only black-box testing is applicable at validation and system level testing.

C. Testing Approaches of Different Paradigm

Conventional testing defined for procedural programs do not fit well in the case of testing an object-oriented program. Conventional software testing tends to focus much on the algorithmic detail of a module and the data that flows across the module interface, whereas object-oriented software tends to focus on the operations that are encapsulated by the class and the state behavior of the class. Several object-oriented features such as data abstraction, inheritance, polymorphism, dynamic binding etc., heavily impact on testing that is not straightforward to make object-oriented systems fit the conventional testing levels. There arises the need for object-oriented testing techniques which suits for object oriented system

Though objects and agents have some similarities, they also differ widely. Agents can be termed as intelligent object as agents possesses certain unique properties such as autonomy, pro-activity, reactivity, social ability etc., Our survey states that the agent oriented software are currently been tested by using the existing object-oriented testing techniques, upon mapping of agent-oriented abstractions into object-oriented constructs. However agent properties such as autonomy, pro-activity, reactivity etc., cannot be mapped into object-oriented constructs. There arises the need for agent-oriented testing techniques for agent-oriented system.

In this paper, various testing approaches applicable for conventional, object-oriented and agent-oriented software are analyzed and compared. We illustrated why conventional testing approach cannot be applied for object-oriented software and also we examined why object-oriented testing approach cannot be applied for agent-oriented software. The paper is organized as follows; Section-2 examines the testing approaches applied for conventional software development. Section-3 examines the testing approaches adopted in object-oriented software development. Section-4 examines the testing approaches of agent-oriented software development and finally section-5 compares the testing approaches of conventional, object-oriented and agent-oriented software system.

II. CONVENTIONAL SOFTWARE TESTING

Conventional method of software development is one of the most popular and widely used approaches. In this method, the common functionalities are grouped together into separate modules which are further divided into several procedures. Conventional software is viewed as algorithm centric where the program is driven by algorithm with data being subsidiary to the program. The phases involved in the conventional software development life cycle are Requirement analysis, Design, Implementation, Testing and Deployment & Maintenance [3]. Testing is an important phase of the software development which plays an important role in ensuring the correctness and the quality of the software. There is plethora of software testing techniques serving different purposes at different phases of software development life cycle [4]. By scope software testing can be categorized into unit testing, integration testing, validation testing and system testing [5].

A. Unit Testing

Unit testing is the primary level of software testing. The testing usually starts with testing the basic entities of the system such as individual program or procedure or function which is termed as a module or component or a Unit. The functionalities of the system are developed and grouped into modules and each module is tested separately to verify if the module satisfies the requirement specification for which it was developed. The module interface is tested to ensure that information flows properly into and out of the program unit under test. The local data structure is examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution. Boundary conditions are tested to ensure that the module operates as specified at the boundaries. Selective testing of execution paths is an essential task during unit test. When data is transferred across modules, there is a possibility of data loss. The data loss may cause adverse effects on the system or the sub functions, when combined may not produce the desired outcome. All independent paths (basis paths) through the control structure are exercised to ensure that all statements in a module have been executed at least once. And finally, all error handling paths are tested.

B. Integration Testing

Integration Testing is a next level of the software testing process where individual unit tested components are combined as dictated by the design and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Since conventional software has hierarchical control structure, the integration may be either top-down or bottom-up approach based on the application. Integration testing can be performed either incrementally or in a non-incremental fashion. Incremental integration testing is done by integrating the units one by one in small increments, thereby errors are easier to isolate and correct. Non-incremental integration testing is performed by integrating all the modules involved in the software as a whole. Generally incremental integration is preferred rather than non-incremental since errors can be easily isolated and corrected when integration proceeded step by step. When two units are to be integrated, few new lines of code need be added for create interaction among units. This new lines of code in turn should not cause new errors. In such case regression testing is applied to ensure that no side effects occur during integration.

C. Validation Testing

Validation testing is done to ensure whether the software reflects the customer requirements. Black-box testing validates the software by deriving appropriate test cases that fully focuses on the functional requirements. Equivalence partitioning and Boundary value analysis are the two important black-box testing approach. Equivalence partitioning divides the input domain of the program into classes. Test cases are derived in such a way that the test data should fall in every equivalence class. Equivalence partitioning strives to define a test case that uncovers classes of errors, thereby reducing the total number of test cases that must be developed. Boundary value analysis is a test case design technique that focuses on boundary values as more possibilities of error occurrences prevail at the boundary conditions.

D. System Testing

System testing is the moment where the success/failure factors and the complexity of the software product are determined. It represents the overall test on assembled software. The main objective of the system testing is to ensure whether the software responds perfectly for all possible input conditions and also handles exception in an acceptable manner. System testing also ensures non-functional requirements of the software by conduction of series of tests such as

1. Performance testing
2. Security testing
3. Recovery testing
4. Stress testing.

Performance testing is designed to test the run-time performance of software within the context of an integrated system.

Security testing attempts to verify that protection mechanisms built into a system will, in fact, protect it from improper penetration.

Recovery testing is a system test that forces the software to fail in a variety of ways and verifies that recovery is properly performed.

Stress testing is designed to confront programs with abnormal situations

III. 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 behavior 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. 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

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 [3]. 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 Integration 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 [7]. 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 [3]. *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.

IV. AGENT-ORIENTED SOFTWARE TESTING

Agent-oriented paradigm can be considered as the extension of object-oriented paradigm [8]. 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 performs parts of its tasks autonomously and to interact with its environment in a useful manner [9]. 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 **anthropomorphic** 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. 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 Integration 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.

C. 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.

D. Testing in Agent-Based System

When object oriented testing methodology is employed to test an agent system, hard and soft goals are identified. The test cases are derived based on these goals. *Random testing* is proposed as an extension of OO testing for agent testing. In this method, one agent is considered at a time and the list of all possible messages which the agent can receive is formulated. The agent is tested by sending random messages in a sequence and the response of the agent corresponding to the message sequence is checked. Agents can also be tested using the *behavior based* technique. This method views each behavior of the agent as a black box. Each behavior can send or receive any number of messages. The test cases must be designed in such a way so as to test the behaviors of the agent by sending messages. The number of test cases may be reduced by employing *partition testing* at agent level. The partition categories can be based on input, output, state, attribute, function, type and behaviors.

Apart from employing the conventional and object oriented methods to test an agent system, a number of agent based testing methods have been propounded. A specialized testing has been introduced for agents developed in Tropos methodology. It is called the goal oriented testing as Tropos is considered to be a *goal*

oriented methodology. The goals are classified according to different perspectives like perform goals, achieve goals and maintain goals according to the agent's attitude towards the goals. It is possible to derive test suites from goal diagrams, by starting from the relationship associated with each goal. Each relationship gives rise to a corresponding test suite, consisting of a set of test cases that are used to check goal fulfillment. These test suites are used to refine goal analysis and detect problem early at the requirement phase. On the later stage, they are executed to test the achievement of the goals from which they were derived.

A special testing strategy has been proposed for agents developed in Prometheus methodology. The focus of this testing method is on identifying the appropriate units and performing automated unit testing. The testing framework includes components that generate the order in which the units are to be tested, generate inputs for creating test cases, automate the test execution, gather results and generate report. There are two types of unit testing available in this testing mechanism. They are *fault-directed testing*, where the intention is to reveal faults in the implementation through failures; and *conformance based testing*, which tests whether the system meets the business requirements. The order of testing is bottom-up, where we test a unit before we test any other unit dependent on it. Plans that form cyclic dependencies are tested together as a single unit.

With the ability of agents to act autonomously, monitoring code changes and generating test cases can be done dynamically. The agent based *regression* testing offers to encompass the regression testing phenomena based on agents. Regression testing is a selective retesting of a software system that has been modified. It is a quality control measure to ensure that the newly modified code sill complies with the specified requirements.

V. CONVENTIONAL VS OBJECT-ORIENTED VS AGENT-ORIENTED SOFTWARE TESTING

A. 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 TESTING

Sl. No	Conventional Testing	Object-Oriented Testing
1.	Module / Subroutine / procedure 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

B. Object-Oriented vs Agent-Oriented Testing

Fig. 2 Kinds of Interactions between objects and between agents

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 [10]. 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 III
DIFFERENCE BETWEEN OBJECT-ORIENTED TESTING AND AGENT-ORIENTED 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 and, 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 created 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. CONCLUSION

Software testing is an important phase of software development process and it is meant for quality assurance and correctness of the software. Testing can be performed at various levels such as unit, integration, validation and system. Agent-oriented paradigm is the recent advancement of software development that acts as a next increment to conventional and object-oriented software development. The paper presents the overview of the testing approaches on various development paradigms such as conventional, object-oriented, agent-oriented and discusses how testing approach differs with respect to different paradigm. Our paper concludes that irrespective of different software development paradigm, the testing levels (unit, integration, validation and system) and testing types (white-box and black-box) are the same but the testing approach concerned to individual paradigm differs widely.

REFERENCES

[1] James.A.Whittaker, "What is software testing? And why is it so hard?", Florida Instittute of Technology,IEEE 2000.

[2] Roger S.Pressman, “Software Engineering – A Practitioner’s Approach”, Tata Mc Graw Hill, 6th edition, 2005.

[3] Walt Scacchi, “Process Models in Software Engineering”, Institute for Software Research, University of California, Irvine, 2001.

[4] V.Therese Clara, Dr. K. Alagarsamy, “A Comparative Study of Traditional Models to Enhance Software Development”, International Journal of Computer Applications, 2010.

[5] Sanjeev Patwa, Dr Anil Kumar Malviya, Dr Narendra Kumar, “Conventional Software Testing:A Discussion”Journal of current computer science and Technology. Vol.1 Issue 8, 2011.

[6] Shalini Gambhir, “ Testing strategies for Object-Oriented Systems” , International Journal of computer Science and Information Technology & Security, Vol.2, No.2, April 2012.

[7] 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.

[8] Nicholas R.Jennings, Michael Woolridge, “Agent oriented software engineering”,Queen Mary and Westfield college, University of London.

[9] Federico Bergenti, Marie Pierre Gleizes, Franco Zambonelli, “Methodologies and Software Engineering for Agent Systems-book”, Kluwer Academic Publishers.

[10] Levine, David, “Relationship between Agent and Object Technologies”, Agent technology green paper, OMG Agent work group, 1999.