iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 6 **Issue:** IV **Month of publication:** April 2018

DOI: http://doi.org/10.22214/ijraset.2018.4728

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 6 Issue IV, April 2018- Available at www.ijraset.com

Software Testing and Development Techniques using UML (Unified Modeling Language)

Vishal Pareek[1], Heena Sheetal[2]
[1, 2]*Department of Computer Science, Faculty of Engineering & Technology, Tantia University*

Abstract: This paper presents the software testing and model based software development with the use of Unified Modeling Language. Software testing is an empirical investigation that is conducted to provide developers and other stakeholders with information about the system under test.
The UML is a tool for specifying software systems that include standardized diagrams to define illustrate and visually map or model a software system's design and structure. This testing process is based on using the modeling techniques that are defined in Unified modeling language. UML structural and behavioral specification diagrams have been used by testing researchers for generation of test scenarios and test data. In software engineering, system modeling is the process of formulating a representation of a real system in an abstract way to understand its behavior. Software testing encourages reusing these models for testing purpose. As object-oriented paradigm becomes more mature, software development industry and researchers turn their attention towards the paradigm.
Keywords: Software testing, UML, Testing tools, Modling techniques, Object Oriented.

I. INTRODUCTION

UML (unified modeling language) becomes the de facto standard for modeling object-oriented software. There are a lot of researches and practices in several areas that effectively utilize uml to achieve better result in works; software testing is among one of them. Lately numerous researchers and practitioners have found the idea of model-based testing using uml very attractive. There is a significant growth of research works and tool implementation in this area during the last couple years.
Nowadays, it is clear that object-oriented paradigm, although provides these profitable features, requires special care for testing.object oriented testing is strategically similar to the testing of conventional system, but is tactically different. Because the object oriented analysis and design models are similar in structure and content to the resultant object oriented program, “testing” begins with the review of these models. A series of tests are designed that exercise class operations and examine whether errors exist as one class operations and examine whether errors exists as one class collaborates with other classes. As classes are integrated to form a subsystem, thread-based, use-based and cluster testing, along with fault-based approaches are applied to fully exercise collaborating classes.
Uml is considered an industry standard modeling language with a rich graphical notation, and comprehensive set of diagrams and elements. It is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software-intensive system under development.[1]

II. UNIFIED MODELING LANGUAGE (UML)

Unified Modeling Language is a standardized general-purpose modeling language in the field of software engineering. The standard is managed, and was created by, the Object Management Group.
UML includes a set of graphical notation techniques to create visual models of software-intensive systems.
The Unified Modeling Language (UML) is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software intensive system under development. UML combines techniques from data modeling (entity relationship diagrams), business modeling (work flows), object modeling, and component modeling. It can be used with all processes, throughout the software development life cycle, and across different implementation technologies. UML has synthesized the notations of the Booch method, the Object-modeling technique (OMT) and Object-oriented software engineering (OOSE) by fusing them into a single, common and widely usable modeling language. UML aims to be a standard modeling language which can model concurrent and distributed systems. UML is a de facto industry standard, and is evolving under the auspices of the Object Management Group (OMG). OMG initially called for information on object-oriented methodologies that might create a rigorous software modeling language. Many industry leaders have responded in earnest to help create the UML standard.[4]

©IJRASET (UGC Approved Journal): All Rights are Reserved

The Unified Modeling Language (UML) is used for helping software analysts and designers to be able to visualize, document and construct object-oriented systems. Three leading advocates of object-oriented methodology, Grady Booch, James Rambaugh and Ivar Jacobson developed UML.

Following are the various modeling diagrams UML notation provides:

A. Use Case Diagram

Use case diagrams give a graphic overview of the actors involved in a system, different functions needed by those actors and how these different functions interact. Use-case diagrams generally show groups of use cases -- either all use cases for the complete system, or a breakout of a particular group of use cases with related functionality It is a list of steps, typically defining interactions between a role (known in UML as an "actor") and a system, to achieve a goal.[9]
.Fig.1shows a use case diagram of a sample point of admission

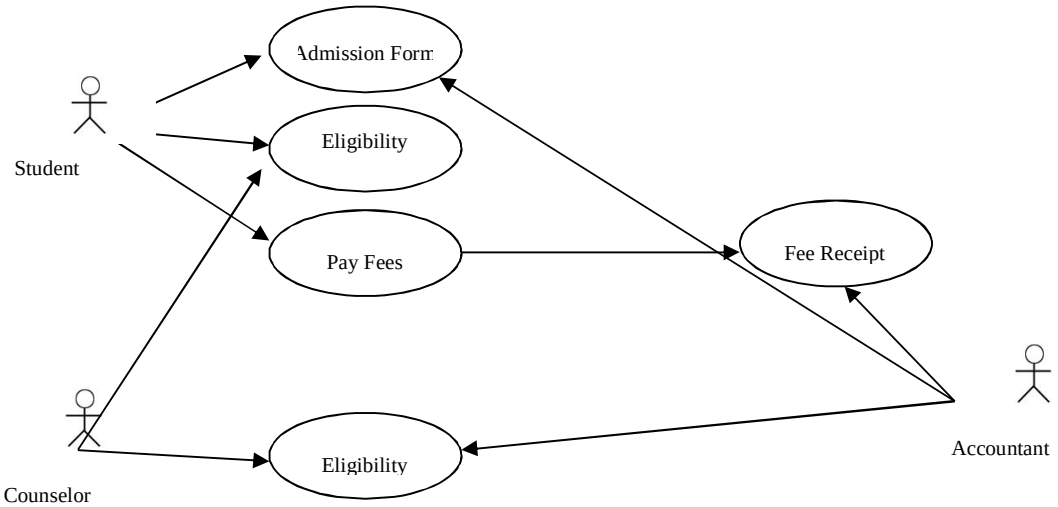


Fig .1 Use Case Diagram of a point of admission

B. Class Diagram

class diagram is represents the basics of object oriented sytem.it show the static structures of system in UML that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes . The class diagram shows how the different entities relate to each other.The main constituents are classes and their relationships. A class is a description of a concept, and may have attributes and operations associated with it. Classes are represented as rectangles**.** [10]

C. Object Diagram

object diagram is an instance of diagram.object diagram represent the static view of system. Sequence diagram:-sequence diagram is an interaction diagram. It depicts how components are wired together to form larger components and or software systems. A sequence diagram has two dimensions: The vertical dimension shows the sequence of messages/calls in the time order that they occur the horizontal dimension shows the object instances to which the messages are sent.[7]

D. Collaboration Diagram

collaboration diagram is an interaction diagram. A Collaboration is a collection of named objects and actors with links connecting them. They collaborate in performing some task.

E. Activity Diagram

Activity diagrams show the procedural flow of control between two or more class objects while processing an activity. Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In UML, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system.[11]

©IJRASET (UGC Approved Journal): All Rights are Reserved

Fig.2 is example of an activity diagram of processing order in a Point of admission system.

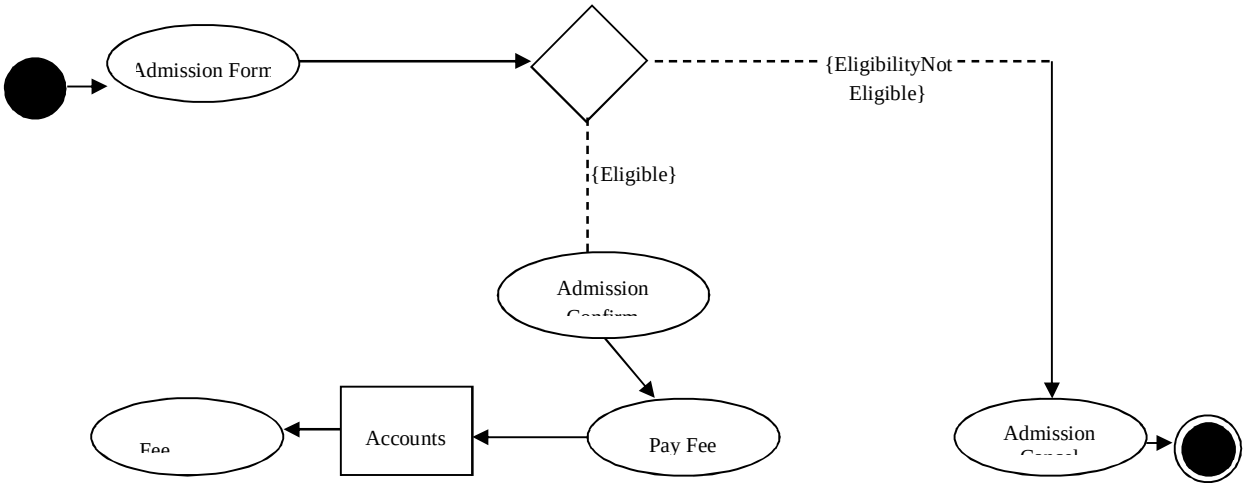


Fig.2 Activity Diagram of a point of admission

F. State chart Diagram

The state chart diagram models the different states that a class can be in and how that class transitions from state to state. States are defined as a condition in which an object exists and it changes when some event is triggered. Its purpose is to model life time of an object from creation to termination.

G. Component Diagram

A component defines its behavior in terms of provided and required interfaces. its provide a physical view of system. It depicts how components are wired together to form larger components and or software systems.

H. Deployment diagram

The deployment diagram shows how a system will be physically deployed. Its purpose is to show where the different components of the system will physically run and how they will communicate with each other.[7]

III. SOFTWARE TESTING

Software testing can also be stated as the process of validating and verifying that a software program

1) Verification: Have we built the software right?

2) Validation: Have we built the right software?

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.

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.[2]

Software testing, depending on the testing method employed, can be implemented at any time in the development process.These two approaches are used to describe the point of view that a test engineer takes when designing test cases. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs.

To cover the strategies and tools associated with object oriented testing

A. Class Testings

Class testing is also known as unit testing. Smallest testable unit is the encapsulated class. Test each operation as part of a class hierarchy because its class hierarchy defines its context of use. These types of tests are usually written by developers as they work on code (white-box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing alone cannot verify the functionality of a piece of software, but rather is used to assure that the building blocks the software uses work independently of each other.[6]

B. Integration Testing

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Integration testing works to expose defects in the interfaces and interaction between integrated components (modules).

©IJRASET (UGC Approved Journal): All Rights are Reserved

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887
Volume 6 Issue IV, April 2018- Available at www.ijraset.com

Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.[3]

C. Validation Testing

Validation succeeds when software functions in a manner that can be reasonably expected by the customer. Focus on user-visible actions and user-recognizable outputs.

D. System Testing

System testing tests a completely integrated system to verify that it meets its requirements.it verifies that a system is integrated to any external or third party systems defined in the system requirements[3].

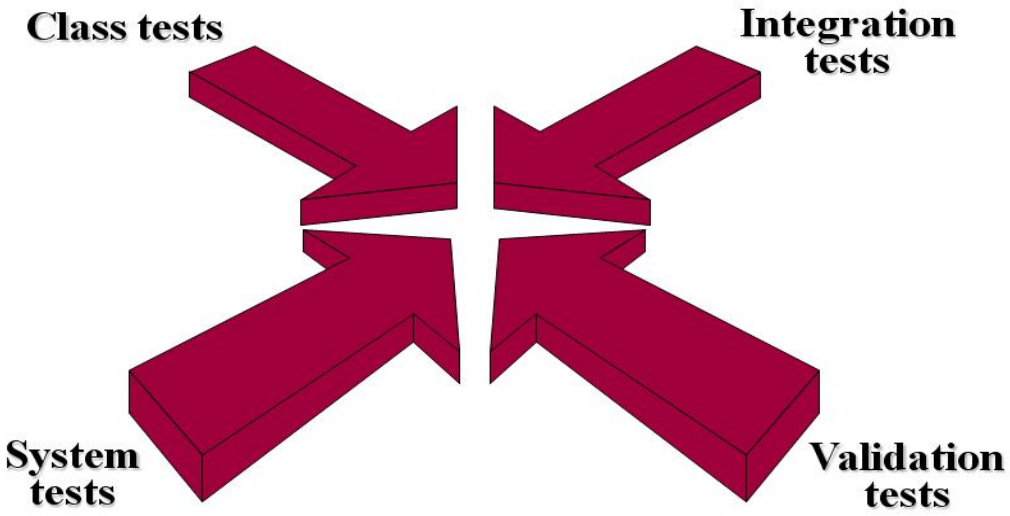


Fig-3.Testing object oriented code

E. Scope Of Software Testing

A primary purpose for testing is to detect software failures so that defects may be uncovered and corrected. This is a non-trivial pursuit. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions. The scope of software testing often includes examination of code as well as execution of that code in various environments and conditions as well as examining the aspects of code: does it do what it is supposed to do and do what it needs to do. In the current culture of software development, a testing organization may be separate from the development team. There are various roles for testing team members. Information derived from software testing may be used to correct the process by which software is developed[6].

F. Functional Vs Non-Functional Testing

Functional testing refers to tests that verify a specific action or function of the code. These are usually found in the code requirements documentation, although some development methodologies work from use cases or user stories. Functional tests tend to answer the question of "can the user do this" or "does this particular feature work".

Non-functional testing refers to aspects of the software that may not be related to a specific function or user action, such as scalability or security. Non-functional testing tends to answer such questions as "how many people can log in at once", or "how easy is it to hack this software".

G. Testing method

Software testing methods are traditionally divided into white- and black-box testing

1) *White box testing:* White box testing is when the tester has access to the internal data structures and algorithms including the code that implement these.

The following types of white box testing :

API testing (application programming interface) - testing of the application using public and private APIs

©IJRASET (UGC Approved Journal): All Rights are Reserved

Code coverage - creating tests to satisfy some criteria of code coverage (e.g., the test designer can create tests to cause all statements in the program to be executed at least once)
Fault injection methods - improving the coverage of a test by introducing faults to test code paths
Mutation testing methods
Static testing - White box testing includes all static testing
Test coverage
White box testing methods can also be used to evaluate the completeness of a test suite that was created with black box testing methods. This allows the software team to examine parts of a system that are rarely tested and ensures that the most important function points have been tested.
Two common forms of code coverage are:

a) Function coverage, which reports on functions executed
b) Statement coverage, which reports on the number of lines executed to complete the test

They both return code coverage metric, measured as a percentage.

2) Black box testing: Black box testing treats the software as a "black box"—without any knowledge of internal implementation. Black box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, fuzz testing, model-based testing, traceability matrix, exploratory testing and specification-based testing. The black box tester has no "bonds" with the code, and a tester's perception is very simple: a code must have bugs. Using the principle, "Ask and you shall receive," black box testers find bugs where programmers do not. But, on the other hand, black box testing has been said to be "like a walk in a dark labyrinth without a flashlight," because the tester doesn't know how the software being tested was actually constructed. [8]

3) Grey box testing: Grey box testing (American spelling: gray box testing) involves having knowledge of internal data structures and algorithms for purposes of designing the test cases, but testing at the user, or black-box level. Manipulating input data and formatting output do not qualify as grey box, because the input and output are clearly outside of the "black-box" that we are calling the system under test. This distinction is particularly important when conducting integration testing between two modules of code written by two different developers, where only the interfaces are exposed for test. However, modifying a data repository does qualify as grey box, as the user would not normally be able to change the data outside of the system under test. Grey box testing may also include reverse engineering to determine, for instance, boundary values or error messages.[5]

IV. CONCLUSIONS

UML diagrams include the use case diagram, class diagram, sequence diagram, statechart diagram, activity diagram, component diagram, and deployment diagram. In general, UML analysis modeling focuses on the user model and structural model and structural model views of the system. UML design modeling addresses the behavioral model, implementation model and environmental model views .
We don't have Object Oriented Testing Model into UML. That's why users adopt different testing model. This solution will help to make a common testing model. The requirement of the user will be that they should be using UML. This paper outlined the use of Unified Modeling Language (UML) as a standard notation of real-world objects in developing object-oriented design methodology for computer applications.

V. ACKNOWLEDGMENT

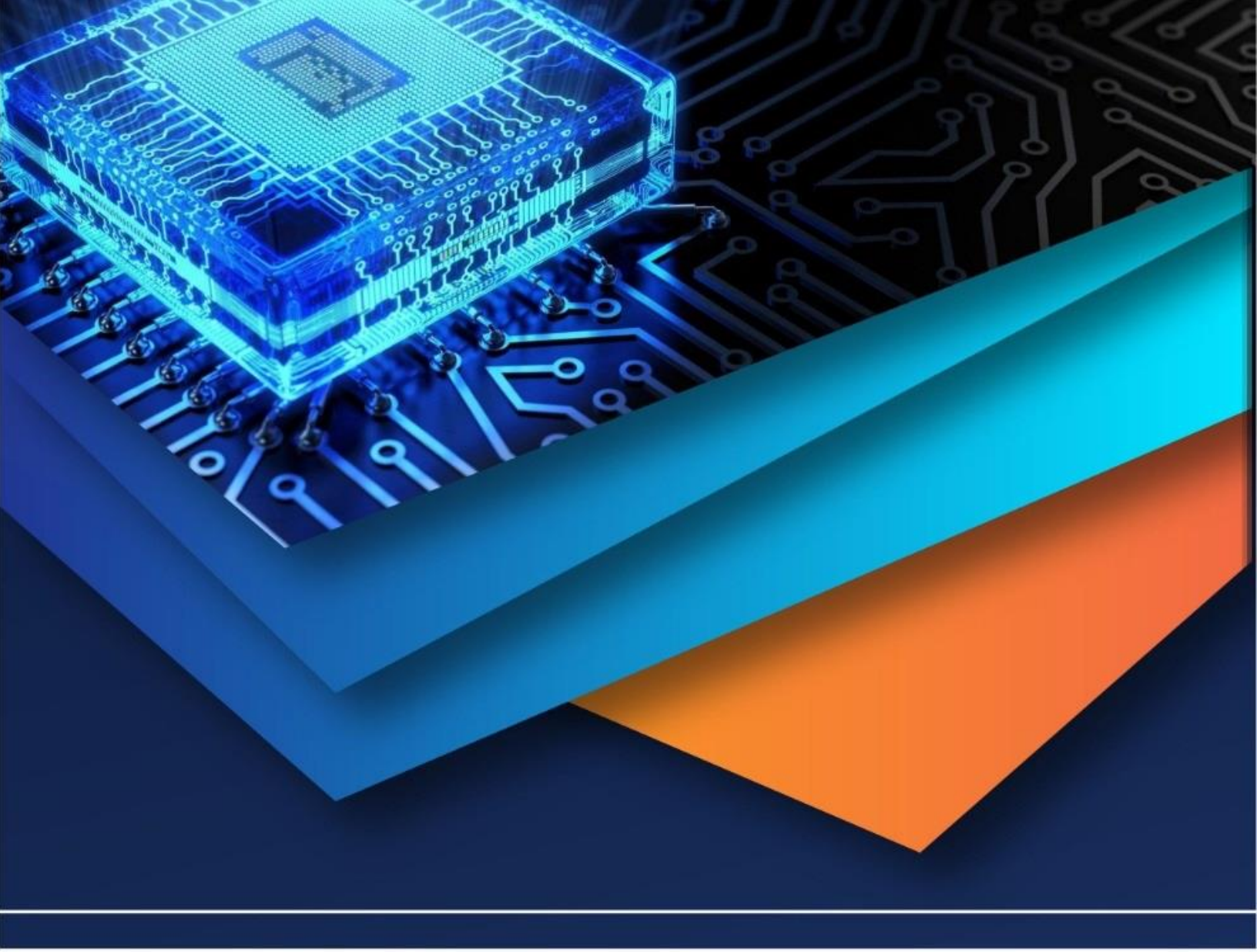
I am grateful to numerous local and global "peers" who have contributed towards shaping for this research and thanks to the God, the Almighty, for His showers of blessings throughout my research work to complete the research successfully.
I would like to express my appreciation and sincere gratitude to my research supervisor ,Dr.Vishal Pareek Ph.D for giving me the opportunity to do research and providing invaluable guidance thoughtful for research .His observations and comments helped me to establish the overall direction of the research and to move forward with investigation in depth. I thank him for providing me with the opportunity to work with a talented team of researchers I am extremely grateful to my parents for their love, prayers, caring and sacrifices for educating and preparing me for my future. Also I express my thanks to my sisters, brother, sister in law and brother in laws for their support and valuable prayers.
I would like to say thanks to my friends and research colleagues ,Tushar Bhatia,Simarjeet Kaur and Manpreet kaur for their constant encouragement. I also thanks all the staff of Research section of Tantia University,Sri Ganganagar Rajasthan for their kindness. Finally, my thanks go to all the people who have supported me to complete the research work directly or indirectly.

©IJRASET (UGC Approved Journal): All Rights are Reserved

REFERENCES

[1] The Object Management Group, "OMG Unified Modeling Language Version 1.4", http://www.omg.org, September 2001.
[2] Andrew, R. France, S. Ghose, G. Craig. "Test Adequacy Criteria for UML Design Models". Journal of Software Testing, Verification and Reliability , 2003
[3] Sudipto Ghosh, Robert France, Conrad Braganza, Nilesh Kawane. "Test Adequacy Assessment for UML Design Model Testing", 14th International Symposium on Software Reliability Engineering (ISSRE 2003), pp.332-343; ISSN, 2003.
[4] R agarwal & Sinha, A. P. Objectoriented modeling with UML: A study of developers' perceptions. Communications of the ACM ,2003
[5] Shivkumar Hasmukhrai Trivedi, "Software Testing Techniques", International Journal of Advanced Research in Computer Science and Software Engineering, Volume 2, Issue 10, October 2012
[6] Howard Foster, Antonia Bertolino, J. Jenny Li, "Automation of Software Test", 6th International Workshop, 2011
[7] Bell D, UML basics: An introduction to the Unified Modeling Language, IBM Developer Works, http://www.ibm.com/developerworks/rational/library/769.html, (2003) June 15.
[8] A. L. Souter and L. L. Pollock, "OMEN: A Strategy for Testing Object-Oriented Software", Proceedings of the International Symposium on Software Testing and Analysis, Portland, Oregon, United States, 2000.
[9] Jacobson, M. Christerson, P. Jonsson, G. Overgard " Object-Oriented software engineering : A use case driven approach " Addison-Wesley 1992
[10] PAGE-JONES, M. Fundamentals of ObjectOriented Design in UML. New York, Dorset House Pub,2000
[11] F. Leymann and D. Roller. Production Workflow: Concepts and Techniques. Prentice Hall, Upper Saddle River, NJ, USA, 2000.

©IJRASET (UGC Approved Journal): All Rights are Reserved



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)