Association between different types of Testing Method using Absolute Architecture

Lalji Prasad[1], Sarita Singh Bhadauria[2]

[1]TRUBA College of Engineering & Technology/ Computer Science &Engineering, INDORE, INDIA

[2]MITS /Department of Electronics, GWALIOR, INDIA

Abstract

software development life cycle testing are one of the most important phase in any software development cycle ,which are assist for produce more reliable and quality software. In this paper proposed a Absolute Architecture Testing Tool (AATT) for testing and try to show that how one testing method correlate with other testing method using ***ArgoUml*** *tool ,this architecture are used in software development testing life cycle which clearly define how one testing method related each other with object oriented perspective and definitely save our time and money*

Keyword*: Software Architecture, Object oriented concept, object oriented testing concept, Software Application Architecture.*

1. Introduction

Software architectures [1] are becoming centric to the development of quality software systems, being the first concrete model of the software system and the base to guide the implementation of software systems. The software architecture of a program or computing system is a depiction of the system that aids in the understanding of how the system will behave. It serves as the blueprint for both the system and the project developing it, defining the work assignments that must be carried out by design and implementation teams. The architecture is the primary carrier of system qualities such as performance, modifiability, and security, none of which can be achieved without a unifying architectural vision. Architecture is an artifact for early analysis to make sure that a design approach will yield an acceptable system.

Software application architecture is the process of defining a structured solution that meets all of the technical and operational requirements, while optimizing common quality attributes such as performance, security, and manageability. It involves a series of decisions based on a wide range of factors, and each of these decisions can have considerable impact on the quality, performance, maintainability, and overall success of the application. Philippe Kruchten, Grady Booch, Kurt

Bittner, and Rich Reitman derived and refined a definition and principle of architecture based on work by Mary Shaw and David Garlan (Shaw and Garlan 1996) [22]. Their definition is:

"Software architecture encompasses the set of significant decisions about the organization of a software system including the selection of the structural elements and their interfaces by which the system is composed; behavior as specified in collaboration among those elements; composition of these structural and behavioral elements into larger subsystems; and an architectural style that guides this organization. Software architecture also involves functionality, usability, resilience, performance, reuse, comprehensibility, economic and technology constraints, tradeoffs and aesthetic concerns".

According to IEEE "Architecture is the fundamental ***organization***[1] *of a* ***system***[2] *embodied in its* ***components***[3]*, their* ***relationships***[3] *to each other, and to the* ***environment***[4]*, and the principles guiding its design and evolution". [3]*

1. A ***system*** *is a collection of components organized to accomplish a specific function or set of functions. 2.The* ***environment****, or context, determines the setting and circumstances of developmental, operational, political, and other influences upon that system. 3. A* ***mission*** *is a use or operation for which a system is intended by one or more* ***stakeholders*** *to meet some set of objectives.4. A* ***stakeholder*** *is an individual, team, or organization (or classes thereof) with interests in, or concerns relative to, a system.*

In this research work emphasis on design architecture for testing based on object oriented perception for software development, here testing classified in three parts: Fault based Testing and Scenario based Testing, Integration testing and its derived classes, Functional Testing, and Class based Testing and its derived classes. Absolute architecture testing tool (AATT), show how one testing technique related to another testing technique, using UML diagram on the based there functionality. Absolute architecture testing tool (AATT), generates incidence metrics of architectures and uses these formalisms to generate appropriate test cases to satisfy the testing criteria.

1. Literature survey

We first develop an intuition for software architecture by appealing to several well-established architectural disciplines. On the basis of this intuition, we present a model of software architecture that consists of three components: elements, form, and rationale. Elements are processing, data, or connecting elements. Form is defined in terms of the properties of, and the relationships among, the elements that is, the constraints on the elements. The rationale provides the underlying basis for the architecture in terms of the system constraints, which most often derive from the system requirements. We discuss the components of the model in the context of both architectures and architectural styles and present an extended example to illustrate some important architecture and style considerations. We conclude by presenting some of the benefits

of our approach to software architecture, summarizing our contributions, and relating our approach to other current work [2]

Generally, we see three major stages in the research and development of testing techniques, each with a different trend. By trend, we mean how mainstream of research and development activities find the problems to solve and how they solve the problems. As described below, technology evolution involves testing technique technologies. The technique used for selecting test data has progressed from an ad hoc approach, through an implementation-based phase, and is now specification based. The literature survey includes the solution approaches of various research studies that dealt with problems related to testing methods and issues in the design of testing tools for various circumstances.

Bertolino and his colleagues use formal architectural descriptions (CHAM) to model the behaviors of interest of the systems. A graph of all the possible behaviors of the system in terms of the interactions between its components is derived and further reduced. A suitable set of reduced graphs highlights the specific architectural properties of the system, and can be used for the generation of integration tests according to a coverage strategy, analogous to the control and data flow graphs in structural testing [6]. Howden W. E. has suggested on 80^{th}, the usual practice of functional testing is to identify functions that are implemented by a system or program from requirement specifications. In this paper, the necessity of design testing and requirement functions is discussed. The paper indicates how systematic design methods, such as structured design and the Jackson design, can be used to construct functional tests. Structured design can be used to identify the design functions that must be tested in the code, while the Jackson method can be used to identify the types of data that should be used to construct tests for those functions [12].

Marciniak proposed a review of test generators is provided in which the major types of test case generators are given and their intended purpose and principles are discussed. A review of the testing process is given in which the entire process of testing is discussed from planning to execution to achieving to maintenance retesting. All of the common terms and ideas are discussed. A review of testing tools is provided in which the testing tool for each purpose is discussed and several state-of-the-art systems are described [13].Richardson D., O'Malley O. and Title C., proposes one of the earliest approaches focusing on utilizing specifications in selecting test cases. In traditional specification-based functional testing, test cases are selected by hand based on a requirement specification, which means functional testing merely includes heuristic criteria. Structural testing has an advantage in that the applications can be automated and the satisfaction determined. The authors propose approaches to specification-based testing by extending a wide variety of implementation-based testing techniques to formal specification languages, and they demonstrate these approaches for the Anna and Larch specification languages [16].

Poston has summarized their work-Integration of all the data across tools, repositories and Integration of control across the tools, this Integration to provide a single graphical interface for the test tool set but has some limitation, it emphasizes only integration tools (usability and portability) [18].Rosenberg and lalji has suggested the approach to software metrics for object-oriented programs must be different from the standard metric sets. Some metrics, such as line of code and cyclomatic complexity, have become accepted as standard for traditional functional/procedural programs. However, for object-oriented scenarios, there are many proposed object-oriented metrics in the literature but has some limitation is that provides only a conceptual framework for measurement [17][21. Anderson emphasize that the software industry has performed a significant amount of research on improving software quality using software tools and metrics that will improve the software quality and reduce the overall development time. Good-quality code will also be easier to write, understand, maintain and upgrade [4]. Lalji and his collogues proposed a full featured comprehensive tool was proposed using the object oriented methodology based architecture [15]. Agrawal has suggested the importance of software measurement is increasing which is leading to the development of new measurement techniques, but has some limitation, in this research paper , object-oriented metrics does not provide any relationship between requirements and testing attributes and object-oriented metrics cannot be evaluated for large data sets[5].

Hartmann proposed a work Unified Modeling Language (UML) is widely used for the design and implementation of distributed, component-based applications, the issue of testing components by integrating test generation and test execution technology with commercial UML modeling tools such as Rational Rose is addressed. The authors present their approach to modeling components and interactions and describe how test cases are derived from these component models and then executed to verify their conformant behavior. The TnT environment of Siemens is used to evaluate the approach by examples [10]. Briand shows that the relationships between most of the existing coupling and cohesion measures for object-oriented (OO) systems and the fault proneness of object-oriented system classes can be studied empirically, but has some limitation Only emphasizes cohesion and coupling metrics [7]. Bitman proposed his research work defines a key problem in software development: changing software development complexity and the method to reduce complexity but has limitation it provides only a complexity measurement technique [8].

Harrison and his colleagues Coupling is the degree of interdependence between two modules. In a good design coupling is kept to a minimum. Coupling should be low in a large and complex system. No coupling is highly desirable, but this is not possible in practice. The strengths and weaknesses of different types of coupling are discussed but have limitation only cohesion and coupling metrics are emphasized [11]. The coupling between the object (CBO) metric of Chidambaram and Kemerer are evaluated for five object-oriented systems and compared with an

alternative design metric called NAS that measures the number of associations between a class and its peers (Harrison R.S). The NAS metric is directly collectible from design documents, such as the object model, but has some limitation No relationship between requirements and testing attributes is provided. A basic idea of the size and effort estimation is not provided, and measuring the complexity of a class is subject to bias [9].

3. Artifacts of Class Testing

In this section, we refer to several of the attributes of object-oriented systems and discuss the different testing techniques suitable for object-oriented software systems. Attributes play an important role in making object-oriented software [20].

a) Encapsulation
Wrapping data and functions into a single unit is known as encapsulation. This restricts the visibility of object states and restricts the observability of intermediate test results. Fault discovery is more difficult in this case.

b) Inheritance
The mechanism of deriving a new class from an old one is called inheritance. The old class is referred to as the base class, and the new one is called the derived class or the subclass. Inheritance results in invisible dependencies between super/sub-classes. Inheritance results in reduced code redundancy, which results in increased code dependencies. If the function is erroneous in the base class, it will also be inherited in the derived class. A subclass cannot be tested without its super-classes. Abstract classes cannot be tested at all.

c) Polymorphism
Polymorphism is one of the crucial features of OOP. It simply means that one name represents multiple forms. Because of polymorphism, all possible bindings must be tested. All potential execution paths and potential errors must be tested. Testing begins by evaluating the OOA and OOD models. Object-oriented analysis models can be tested using the collected requirements and use cases. Object-oriented design can be tested by using the class and sequence diagrams. Structured walkthroughs and reviews should be conducted to ensure correctness, completeness and consistency.

Object–oriented programming is centered on concepts such as Object, Class, Message, Interfaces, Inheritance, and Polymorphism. Traditional testing techniques can be adopted in object-oriented environments by using the following techniques: Function-based testing, Class testing, Integration testing, Fault-based testing, and Scenario-based testing [19].

Generally, three major stages in the research and development of testing techniques have been seen, each with a different trend. By trend, it is meant how mainstream of research and development activities find the problems to solve and how they solve the problems. As described below, technology evolution involves testing technique technologies. The technique used for selecting test data has progressed from an ad hoc approach, through an implementation-based phase, and is now specification based. The literature survey includes the solution approaches of various research studies that dealt with problems related to testing methods and issues in the design of testing tools for various circumstances.

4. Relationship between different types of testing method

Following relationship depicts how one testing method associates each other through possible object- oriented concepts.

a) Fault based Testing and Scenario based Testing:

The first category consists of fault-based testing and scenario-based testing.

- The objective of fault-based testing within an OO system is to design tests that have a high likelihood of uncovering plausible faults. Because the product or system must conform to customer requirements, the preliminary planning required to perform fault-based testing begins with the analysis model. The tester looks for possible faults (i.e., aspects of the implementation of the system that may result in defects).

- When errors associated with incorrect specifications occur, the product does not do what the customer wants. Scenario-based testing concentrates on what the user does, not what the product does. This means capturing the tasks (via use-cases) that the user has to perform, then applying them and their variants as tests. Scenarios uncover interaction errors. Scenario-based testing tends to exercise multiple subsystems in a single test.

Relationships:

a) Uni-aggregation relationship between scenario based testing and fault based testing:
Fault based testing is a part of scenario based testing and therefore a uni- aggregation relationship exists between these two testing's.

b) Composition relationship between scenario based testing and use based testing:
Since, to perform scenario based testing use cases are designed and tested. Thus, use case based testing acts as a part of scenario based testing. Therefore, composition relationship exists between scenario based and use based testing.

b) Integration Testing and its derived classes:

In the second category, integration testing is further divided into three parts: Thread based testing; cluster 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.

- Use-based testing begins the construction of the system by testing those classes (called independent classes) that use very few (if any) server classes. After the independent classes are tested, the next layers of classes, called dependent classes, that use the independent classes are tested. This sequence of testing layers of dependent classes continues until the entire system is constructed.

- Cluster testing is one step in the integration testing of OO software. Here, a cluster of collaborating classes is determined by examining the Class Responsibility Collaboration(CRC) and object-relationship model.

Relationships:

a) Generalization/Inheritance relationship between integration testing and threads based testing, cluster based testing and use based testing:
Since, Integration testing is further divided into three parts-threads, cluster and use-based testing and therefore possess generalization relationship as integration testing class is generalized class for thread based, cluster based and use based testing.

b) Aggregation relationship between thread based testing and cluster based testing:
A cluster of collaborating classes have several threads in it and thus threads are part of clusters. But, existence of threads may exist independent of the clusters. Hence, aggregation which is one of the forms of association relationship exists between cluster and thread based testing.

c) Bidirectional navigability between Use based testing and thread based testing:
Use based testing begins the construction of the system by testing dependent and independent classes having threads. So testing can be done by designing use case for each thread firstly or after testing the threads, use based testing can be formed. Thus, bidirectional navigability exists between these two testing classes.

c) Functional Testing, Class based Testing and its derived classes:

The third category consists of functional testing, class based testing and its derived classes. This category is directly based on the requirements and specifications of software products. Partitioning-based testing and random testing are derived from class based testing and uses some properties of class based testing. Partition based testing is further

classified into three types: State based testing, Attribute based testing and Category based testing.

Relationships:

a) Interdependency between functional testing and class based testing:
Inter dependence relationship exists between functional testing and class based testing as when functional specification are input for function level testing of any testing tools. Accordingly, functional specifications construct class based testing.

b) Generalization/Inheritance relationship between class based testing and partition & random based testing:
Class based testing is divided into two parts: partition based class testing and random based testing. Thus, their exists the generalization relationship as partition based class testing and random based testing are derived from class based testing and uses some properties of class based testing.

c) Generalization/Inheritance relationship between partition based testing and state based, attribute based and category based testing:
As partition based testing is further classified into state based, attribute based and category based testing and therefore, partition class has a generalization relationship with the other three testing classes respectively.

d) Aggregation relationship between state based testing and attribute based testing:
State based testing is associated with attribute based testing with its special form called as aggregation as each attribute possess some state and to perform attribute based testing, state based testing should be performed.

e) Composition relationship between category based testing and state based testing:
Category based testing possess composition relationship with the state based testing as for a particular category of a software testing, various objects may have various states and thus required to be tested based on their states.

f) Composition relationship between partition based testing and state based, attribute based and category based testing:
Since, state based, attribute based and category based testing can only be performed after partitioning the class. Thus, composition relationship exists between them as state based, attribute based and category based testing cannot exists without partition based testing.

g) Multiplicity relationship between partition based testing and state based, attribute based and category based testing:
Multiplicity between partition based testing and state based testing denotes that at least 0 or 1 object of partition based testing is related with 1 or more objects of state based testing.

5. Conclusion:

This architecture tool facilitate in software testing life cycle for deciding relationship of between of testing technique. This tool, help developers and testing community for determine software quality in less time and less cost and improves quality of software, but this paper emphasis only, conceptual framework for absolute architecture. Future extension of this work, describe each testing technique through case-study and determine software quality by using metrics measurement for large data set.

6. References:

[1]. D. Garlan. Software Architecture: a Roadmap. In *ACM ICSE 2000, The Future of Software Engineering*, pages 91–101. A. Finkelstein, 2000.

[2]. Dewayne E. perry, Alexander L.Wolf, Foundations for the study of software architecture, ACM SIGSOFT Software Engineering Notes, Volume 17 Issue 4, Oct. 1992

[3]. IEEE Standard 1471 Recommended Practice for Architectural Description of Software-Intensive System, publishes in Oct.2000.

[4]. Anderson John L. Jr., "How to Produce Better Quality Test Software", IEEE Instrumentation & Measurement Magazine, August 2005.1

[5]. Agarwal K. K., Sinha Y., Kaur A. and Malhotra R.," Exploring Relationships among coupling metrics in object oriented systems. Journal of CSI vol. 37, no.1, January March 2007.2

[6]. Bertolino A., Inverardi `P., Muccini H. and Rosetti A., "An approach to integration testing based on architectural descriptions," Proceedings of the IEEE ICECCS- 97, pp. 77-84.3

[7]. Briand Lionel C. and Daly J., "A Comprehensive Empirical Validation of Design Measures for Object-Oriented Systems", Fraunhfer IESE, 1999.6

[8]. Bitman William R., " Balancing software composition & inheritance to improve reusability cost & error rate", Johns Hopkins APL Technical Digest, Volume 18 November 1997.8

[9]. Chidamber S. and Kemerer C., "A metrics suite for object oriented design", IEEE Trans. Software Eng., vol. 20, pp. 476-493, 1994.9

[10]. Hartmann J., Imoberdorf C. and Meisinger M., "UML-Based Integration Testing," Proceedings of the International Symposium on Software Testing and Analysis, ACM SIGSOFT Software Engineering Notes, August 2000.13

[11]. Harrison R., Counsell S. and Nithi R., "Coupling metrics for object oriented design", Software metrics, symposium, MD, USA, 19 November 1998.15

[12]. Howden W. E., "Functional Testing and Design Abstractions", the Journal of System and Software, Volume 1, 1980, pp. 307-313.17

[13]. Marciniak J. J., "Encyclopedia of software engineering", Volume 2, New York, NY: Wiley, 1994, pp.1327-1358.23

[14]. Pressman Roger S., "Software Engineering – A Practitioner's Approach" McGraw Hill International Edition sixth, 2004.24

[15]. Prasad Lalji. Bhadauria S. and Kothari A., "A full featured component object oriented based architecture testing tool" ,IJCSI Sep2011.25

[16]. Richardson D., O'Malley O. and Title C., "Approaches to specification-based testing", ACM SIGSOFT Software Engineering Notes, Volume 14 , Issue 9, 1989, pp. 86 – 96.26

[17]. Rosenberg Linda H., "Applying & interpreting object oriented Metrics", 2008.27

[18]. Robert M. Poston, "Testing tool combines best of new and old", IEEE Software. March 2005.28

[19]. Suganya G. and Neduncheliyan S., A Study of Object Oriented Testing Techniques: Survey and Challenges. IEEE Feb. 2010, pp: 1 – 5.31

[20]. Prasad Lalji ,Rashmi y. Abhay K., Measurement of Software Using Various Construct in Information Model Proceedings of the International Conference on Advances in Computing, Communication and Control (ACM Digital Library),1909, ISBN: 978-1-60558-351-8.

[21]. Prasad Lalji,Aditi nagar," Experimental Analysis of Different Metrics (object –oriented and structural) of software, First International Conference on Computational Intelligence,

Communication Systems and Networks IEEE Computer Society Washington, DC, USA ,2009, ISBN: 978-0-7695-3743-6.

[22].http://msdn.microsoft.com/en-us/library/ee658098.aspx