A full featured component (object oriented) based architecture testing tool

Sarita Singh Bhadauria [1], Abhay Kothari[2] and Lalji Prasad[3]

[1] **MITS /Department of Electronics,
GWALIOR, INDIA**

[2] **IIST/ Computer Engineering,
INDORE, INDIA**

[3] **Truba College of Engineering & Technology/ Computer Engineering,
INDORE, INDIA**

Abstract
Object-orientation has rapidly become accepted as the preferred paradigm for large-scale system design. The product created during Software Development effort has to be tested since bugs may get introduced during its development. In this research work we 1) establish a requirement specification for a comprehensive software testing tool. 2) This will involve studying the feature set offered by existing software testing tools and their limitations. This will be able to overcome the limitations of limited feature sets of existing software tools. 3) To propose a comprehensive architecture of a software testing tool, this will include most of the features required for a software testing tool. 4) The purpose is to avoid compatibility problems which are incurred by interfacing various tools to utilize individual tools strengths. Also, as different tools are having different user interfaces, it takes effort to learn, how to use them. A full featured, comprehensive tool is a solution to all of these problems. We intend to propose the object oriented methodology based architectures for the comprehensive tool.
Keywords: Fault-based Testing, Scenario-based Testing, comprehensive software testing tool, Compatibility problem

1. Introduction

The testing of software is an important means of assessing the software to determine its quality. Since testing often consumes 40~50% of development efforts, and consumes more effort for systems that require higher levels of reliability, it is a significant part of the software engineering. With the development of Fourth generation languages (4GL), which speeds up the implementation process, the proportion of time devoted to testing increased. As the amount of maintenance and upgrade of existing systems grow, significant amount of testing will also be needed to verify systems after changes are made [2]. Definition of testing:
"Program testing is a rapidly maturing area within software engineering that is receiving increasing notice both by computer science theoreticians and practitioners. Its general aim is to affirm the quality of software systems by systematically exercising the software in carefully controlled circumstances."

E. Miller \Introduction to Software Testing Technology"[1]

The remainder of the paper is organized as follows: Section 2 gives an idea major stage of research & Literature survey of related to Object Oriented Software and presents the various stages of testing. Section 3 definition of object-oriented testing. Section 4 presents the specialized techniques available for Object Oriented environment and architecture object oriented testing Section 5 present objective of this Research Section 6 present Conclusion & Future work .In abstract four features mention three feature cover here last features cover in my next research paper.

2. The Major Stages of Research & Development Trends (Literature survey)

Generally, we see three major stages of the research and development of testing techniques, each with a different trend. By trend we mean the how mainstream of research and development activities find the problems to solve, and how they solve the problems. As below given "Technology Evolution" , testing technique technologies, thus the ways of selecting test data have developed from ad hoc, experienced implementation-based phase, and is focusing on specification-based now.

1950 – 1970: Ad Hoc
From the years 1950 and 1970, there were few research results on testing techniques except for the conceptual ideas of testing goals. It's possible that research results before 1970 are too old to be in the reach of current bibliography collections. To avoid being influenced by this factor, we looked at many testing survey papers in the 1980s, which should have had the "ancient" studies in hand by the time they performed their study. We suppose their surveys at least addressed the most important technical contributes before their time, and we can build our research for the decades before 1970 on theirs. Based on above assumption, we define the period

between 1950 a nd 1970 a s being ad hoc. During this period, major research interest focuses on the goal of testing, and there are quite a few discussions on how to evaluate if a test is good. Meanwhile, testing had become gradually independent from part of debugging activities, to a necessary way to demonstrate that a program satisfies its requirements, as is seen in the GH88 model. At the same time, if we look from Shaw's view, we can see that the whole world of software engineering was in its programming-in-any-which-way stage. It's very natural that testing stayed in its ad hoc stage, where test data is selected randomly and in an unorganized, undirected way.

1971 – 1985: Emphasize on Implementation (for small program)

Beginning from the mid 1960s to the mid 1980s, the whole software engineering research community shifted it paradigms to the program-in-the-small stage, and then started the program-in-the-large stage. he main changes this migration brought to software development were that the characteristic problems changed from small programs, to larger programs and algorithms, and were on the way to developing more complex problems. In response to this significant change, researches on testing techniques began their prosperity. On the structural side, in 1975, 1976, 1980, and 1985, although the whole software engineering community was facing the challenge of switching the gear of developing from comparably simple programs to complex large systems, it took time for testing community to react to the change, specifically, in approximately 5 years. From the figure we also find that only one significant result for functional testing appeared in this period. The reason is obvious. Functional testing is based on requirements and has consisted merely of heuristic criteria. It is difficult to determine when and if such criteria are satisfied without being able to express the requirements in an efficient, rigorous, unambiguous way. This was in part the motivation for developing implementation-based testing techniques; they have the advantage that their application can be automated and their satisfaction determined. Fortunately the research appeared during this period set up a very good tone of successive researches, since it moved emphasis from the simple input/output specifications that testers often used in this period to a higher level – the design of the system. In this period, how to test a "program", instead of a "system", still drew the attention of researchers and practitioners. However the whole software engineering had begun to get ready for moving from the stage of programming-in-the-large to a higher level.

1986 – Current: Emphasize on Specification and System

As software become more and more pervasive, the engineering for this area experienced the shift from programming-in-the-large to programming-in-the-world, starting from the mid 1980s. The characteristic problems changed from algorithms, to system structures, and component interfaces. Systems have been specified in more complex ways. Studies in software architecture and formal methods have brought a lot of facilities as well as inspiration to the way people specifying their systems. Based on these studies, software system now can be specified in more rigorous, understandable, automatable ways, which has brought great chances to improve functional testing techniques. Meanwhile, software development is no longer limited to standalone systems, in reality, there have been more and more needs to develop distributed, object-oriented, and component based systems. The researchers in testing community have responded this trend and move their emphasis accordingly. Starting from the late 1980s, many researchers have made use of the achievements of formal methods and logical analysis. There is still limitation in the specification capabilities so that researchers have been calling for better specification methods to improve their results. Both functional and structural testing techniques have benefited from the enhancement of software specification technologies. The widespread developing and using of object-oriented technologies, COTS software and component based systems has brought a great density of testing researches on these kinds of systems. The earliest OO testing studies appeared in the early 1990s. Most of them use traditional functional and/or structural techniques on the components, i.e. classes and so on. Researchers have proposed new problems and solutions on testing the connections and inheritances among components. Both structural and functional techniques are hired in their approaches, and it has proven to be an effective method to integrate the two techniques for testing complex systems.

2.1. Literature Survey

[1] G. Bernet, L. Bouaziz, and P. LeGall, "A Theory of Probabilistic Functional Testing," Proceedings of the 1997 International Conference on Software Engineering, 1997, pp. 216 –226

[BBL97] A framework for probabilistic functional testing is proposed in this paper. The authors introduce the formulation of the testing activity, which guarantees a certain level of confidence into the correctness of the system under test. They also explain how one can generate appropriate distributions for data domains including most common domains such as intervals of integers, unions, Cartesian products, and inductively defined sets. A tool assisting test case generation according to this theory is proposed. The method is illustrated on a small formal specification.

[2] B. Beizer, "Software Testing Techniques," Second Edition, Van Nostrand Reinhold Company Limited,1990, ISBN 0-442-20672-0

[Beizer90] This book gives a fairly comprehensive overview of software testing that emphasizes formal models for testing. The author gives a general overview of the testing process and the reasons and goals for

testing. In the second chapter of this book, the author classifies the different types of bugs that could arise in program development. The notion of path testing, transaction flow graphs, data-flow testing, domain testing, and logic-based testing are introduced in detail in the chapters followed. The author also introduces several attempts to quantify program complexity, and more abstract discussion involving paths, regular expression, and syntax testing. How to implement software testing based on the strategies is also discussed in the book.

[3] S. Beydeda and V. Gruhn, "An integrated testing technique for component-based software," ACS/IEEE International Conference on Computer Systems and Applications, June 2001, pp 328 – 334
[BG01] Testing is made complicated with features, such as the absence of component source code, that are specific to component-based software. The paper proposes a technique combining both black-box and white-box strategies. A graphical representation of component software, called component-based software flow graph (CBSFG), which visualizes information gathered from both specification and implementation, is described. It can then be used for test case identification based on well-known structural techniques.

[4] A. Bertolino, P. Inverardi, H. Muccini, and A. Rosetti, "An approach to integration testing based on architectural descriptions," Proceedings of the IEEE ICECCS- 97, pp. 77-84
[BIMR97] In this paper the authors propose to use formal architectural descriptions (CHAM) to model the behavior of interest of the systems. 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 specific architectural properties of the system, and can be used for the generation of integration tests according to a co verage strategy, analogous to the control and data flow graphs in structural testing.

[5] J.B. Good Enough and S. L. Gerhart, "Toward a Theory of Test Data Selection," IEEE Transactions on Software Engineering, June 1975, pp. 156-173
[GG75] This paper is the first published paper, which attempted to provide a theoretical foundation for testing. The "fundamental theorem of testing" brought up by the authors characterizes the properties of a completely effective test selection strategy. The authors think a test selection strategy is completely effective if it is guaranteed to discover any error in a program. As an example, the effectiveness of branch and path testing in discovering errors is compared. The use of decision table (a mixture of requirements and design-based functional testing) as an alternative method is also proposed.

[6] D. Gelperin and B. Hetzel, "The Growth of Software Testing", Communications of the ACM, Volume 31 Issue 6, June 1988, pp. 687-695
[GH88] In this article, the evolution of software test engineering is traced by examining changes in the testing process model and the level of professionalism over the years. Two phase models, the demonstration and destruction models, and two life cycle models, the evolution and prevention models are given to characterize the growth of software testing with time. Based on the models a prevention oriented testing technology is introduced and analyzed in detail.

[7] J. Hartmann, C. Imoberdorf, and M.Meisinger, "UML-Based Integration Testing," Proceedings of the International Symposium on Software Testing and Analysis, ACM SIGSOFT Software Engineering Notes, August 2000
[HIM00] Unified Modeling Language (UML) is widely used for the design and implementation of distributed, component-based applications. In this paper, 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, 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

[8] W. E. Howden, "Reliability of the Path Analysis Testing Strategy", IEEE Transactions on Software Testing, September 1976, pp. 208-215
[Howden76] The reliability of path testing provides an upper bound for the testing of a subset of a program's paths, which is always the case in reality. This paper begins by showing the impossibility of constructing a test strategy that is guaranteed to discover all errors in a program. Three commonly occurring classes of errors, computations, domain, and sub case, are characterized. The reliability properties associated with these errors affect how path testing is defined.

[9] W. E. Howden, "Functional Testing and Design Abstractions," The Journal of System and Software, Volume 1, 1980, pp. 307-313
[Howden80] The usual practice of functional testing is to identify functions that are implemented by a system or program from requirements specifications. In this paper, the necessity of testing design as well as 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 which should be used to construct tests for those functions.

[10] J. C. Huang, "An Approach to Program Testing," ACM Computing Surveys, September 1975, pp.113-128
[Huang75] This paper introduces the basic notions of dynamic testing based on detailed path analysis in which

full knowledge of the contents of the source program being tested is used during the testing process. Instead of the common test criteria by which to have every statement in the program executed at least once, the author suggested and demonstrated by an example, that a better criterion is to require that every edge in the program diagraph be exercised at least once. The process of manipulating a program by inserting probes along each segment in the program is suggested in this paper.

[11] P. Jalote and Y. R. Muralidhara, "A coverage based model for software reliability estimation, "Proceedings of First International Conference on Software Testing, Reliability and Quality Assurance, 1994, pp. 6 –10 (IEEE)
[JM94] There exist many models for estimating and predicting the reliability of software systems, most of which consider a s oftware system as a b lack box and predict the reliability based on the failure data observed during testing. In this paper a reliability model based on the software structure is proposed. The model uses the number of times a particular module is executed as the main input. A software system is modeled as a graph, and the reliability of a node is assumed to be a function of the number of times it gets executed during testing – the larger the number of times a node gets executed, the higher its reliability. The reliability of the software system is then computed through simulation by using the reliabilities of the individual nodes.

[12] J. J. Marciniak, "Encyclopedia of software engineering", Volume 2, New York, NY: Wiley, 1994, pp.1327-1358
[Marciniak94] A book intended for software engineers, this book gives introductions, overviews, and technical outlines of the major areas in software engineering. A review in to test generators is given where the major types of test case generators are given and their intended purpose and principles are discussed. A review on the testing process is given where the entire process of testing is discussed from planning to execution to achieving to maintenance retesting. All the common terms and ideas are discussed. A review of testing tools is given where the testing tool for each purpose is discussed and a co uple for state of the art systems is given.

[13] E. F. Miller, "Introduction to Software Testing Technology," Tutorial: Software Testing & Validation Techniques, Second Edition, IEEE Catalog No. EHO 180-0, pp. 4-16
[Miller81] This article serves as the one of the introductory sections of the book Tutorial: Software Testing & Validation Techniques. A cross section of program testing technology before and around the year 1980 is provided in this book, including the theoretical foundations of testing, tools and techniques for static analysis and dynamic analysis, effectiveness assessment, management and planning, and research and development of soft ware testing and validation. The article briefly summarizes each of the major sections. The article also gives good view of the motivation forces, the philosophy and principles of testing, and the relation of testing to software engineering.

[14] D. Richardson, O. O'Malley and C. Title, "Approaches to specification-based testing", ACM SIGSOFT Software Engineering Notes, Volume 14 , Issue 9, 1989, pp. 86 – 96
[ROT89] This paper 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, thus makes functional testing consist merely heuristic criteria. Structural testing has the advantage of 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 be applicable to formal specification languages, and demonstrate these approaches for the Anna and Larch specification languages.

[15] S. Redwine & W. Riddle, "Software technology maturation," Proceedings of the Eighth International Conference on Software Engineering, May 1985, pp. 189-200
[RR85] In this paper, a variety of software technologies are reviewed. The technology maturation process by which a p iece of technology first gets the idea formulated and preliminarily used, then is developed and extended into a broader solution, and finally is enhanced to product-quality applications and marketed to the public. The time required for a piece of technology to mature is studied, and the actions that can accelerate the maturation process are addressed. This paper serves as a very good framework for technology maturation study.

[16] S. Rapps and E. J. Weyuker, "Selecting Software Test Data Using Data Flow Information," IEEE Transactions on Software Engineering¸ April 1985, pp. 367-375
[RW85] A family of test data selection criteria based on data flow analysis is defined in this paper. The authors contend that data flow criteria are superior to currently path selection criteria being used in that using the latter strategy program errors can go undetected. Definition/use graph is introduced and compared with a program graph based on the same program. The interrelationships between these data flow criteria are also discussed.

[17] M. Shaw, "Prospects for an engineering discipline of software," IEEE Software, November 1990, pp.15-24
[Shaw90] Software engineering is still on its way of being a true engineering discipline. This article studies the model for the evolution of an engineering discipline and applies it to software technology. Five basic steps are suggested to the software profession to take towards

a true engineering discipline: to understand the nature of expertise, to recognize different ways to get information, to encourage routine practice, to expect professional specializations, and to improve the coupling between science and commercial practice. The significant shifts in research attention of software engineering since the 1960s are also given in this article.

[18] L. J. White and E. I. Cohen, "A Domain Strategy for Computer Program Testing," IEEE Transactions on Software Engineering, May 1980, pp. 247-257
[WC80] Domain errors are in the subset of the program input domain, and can be caused by incorrect predicates in branching statements or incorrect computations that affect variables in branching statements. In this paper a set of constraints under which it's possible to reliably detect domain errors is introduced. The paper develops the idea of linearly bounded domains. The practical limitations of the approach are also discussed, of which the most severe is that of generating and then developing test points for all boundary segments of all domains of all program paths.

[19] J. A. Whittaker, "What is Software Testing? And Why Is It So Hard?" IEEE Software, January 2000, pp. 70-79
[Whit00] Being a practical tutorial article, the paper answers questions from developers how bugs escape from testing. Undetected bugs come from executing untested code, difference of the order of executing, combination of untested input values, and untested operating environment. A four-phase approach is described in answering to the questions. By carefully modeling the software's environment, selecting test scenarios, running and evaluating test scenarios, and measuring testing progress, the author offers testers a structure of the problems they want to solve during each phase.

[20] Poston (2005), Williams (2002), Hareton (1998) (Poston, 2005), Robert M. Poston " Testing tool combine best of new and old," IEEE Software. March 2005. (Williams. 2002) Williams et. Al., "The STCL Test Tool Architecture," IBM Systems Journal, Vol. 41, No. 1, 2002. (Hareton, 1998) Hareton K., N. Leung"Test tools for the year 2000 challenges" 1998 IEEE.
Here we summarized their work.

-Integration of all the data across tools and repositories.
- Integration of control across the tools
-Integration to provide a single graphical interface into test tool set.

Limitation: its emphasize only integration tool (usability &portability)

[21].Rosenberg (2008), Dr. Linda H. Rosenberg, "Applying & interpreting object oriented Metrics," 2008.
The approach to software metric for object oriented program must be different from the standard metric sets. Some metrics, such as, line of code & cyclomatic complexity, have became accepted as standard for traditional functional / procedural programs, but for object oriented scenario, there are many proposed object oriented metrics in the literature
Limitation: this provides only conceptual framework for measurement

[22] Agrawal (2007),K. K. Agarwal, Yogesh Sinha, Arvinder Kaur, Ruchika Malhotra " Exploring Relationships among coupling metrics in object oriented systems. Journal of CSI vol. 37, no.1, January March 2007.
As per this paper the importance of software measurement is increasing leading to development of new measurement techniques.
Limitation:

a) It's not provide any relationship between requirement & testing attribute.
b) It cannot evaluate for large data sets.

"Software quality is another focus of our research. Metrics fall into two categories the productivity and the quality. Most of our object oriented metrics are quality related. We wish to achieve good maintainability, reusability, flexibility and portability in the architecture of the software testing tool under construction".

[23] Anderson (2005), John L. Anderson Jr. "How to Produce Better Quality Test Software", IEEE Instrumentation & Measurement Magazine, August 2005.
They emphasize that the software industry has performed a s ignificant amount of research on improving software quality using software tools & metrics will improve the software quality and reduce the overall development time. Good quality code will also be easier to write understand, maintain and upgrade.
Limitation:
a) It's not providing any relationship between requirement testing attribute.
b) Its not provide full featured testing tool (only Complexity & cohesion measure).
c) Here provide only conceptual framework for measurement.

[24] Briand (1999), Lionel C. Briand, John Daly "A Comprehensive Empirical Validation of Design Measures for Object-Oriented Systems", Fraunhfer IESE, 1999.
This paper aims is that empirically the relationships between most of the existing coupling & Cohesion measures for object oriented (OO) system & fault proneness of object oriented system classes can be studied
Limitation: a) Only emphasis on cohesion & coupling metrics

[25] Bitman(1997),William R. Bitman, " Balancing software co mposition & i nheritance t o i mprove reusability co st & error rate," Johns H opkins APL Technical Digest, Volume 18 November 1997.

This research defines a key problem in software development of changing software development complexity and the method to reduce complexity.
Limitation: a) Its provide only complexity measurement technique.

[26] Krauskopf(1990), Harrison(1998), R. Harrison, S. Counsell, R. Nitin, "Coupling metrics for object oriented design," Radical eye software, 1998.(Juan) Juan Carlos Esteva, "Learning to Recognize" (Krauskopf, 1990) Jan Krauskopf, "The cohesive highs and the coupling lows of good software design", IEEE, 1990.

Coupling is the degree of interdependence between two modules. In a good design coupling is kept minimum. Coupling should be low in large and complex system. No coupling is highly is desirable but practically it is not possible. The good & bad points of different types of coupling are discussed
Limitation : a) Only emphasis on cohesion & coupling metrics

[27] The coupling between object (CBO) metric of Chidambaram & kemerer are evaluated for five object oriented systems & compared with alternative design metric called NAS which measure the number of association between class & its peers (Harrison R.S). The NAS metric is directly collectible from design documents such as object model.
Limitation:
a) it's not provide any relationship between requirement & testing attribute.
b) it's not provide some basic idea for size & effort estimation.
c) Measuring complexity of a class is subject to bias.

3. Object Oriented Testing

Within the last decade, the object-oriented paradigm (OOP) has been established as is programming method with great possibilities. Object-orientation has rapidly become accepted as the preferred paradigm for large-scale system design, it has many features. An object is an entity composed of data and procedures. The procedures, referred to as methods, implement the operations on the object's data. Each object **has** a state, an identity, and a behavior. The definition of the type of object is a description of its capabilities. OO testing concentrates on the states of the objects and their interactions. In object orientation testing system classes play important role, classes are the smallest testable unit, its provide an excellent structuring mechanism. They allow a system to be divided into well defined units which may then be implemented separately. Second, classes support information-hiding. A class can export a purely procedural interface and the internal structure of data may be hidden. This allows the structure to be changed without affecting users of the class, thus simplifying maintenance. Third, object-orientation encourages and supports software reuse. This may be achieved either through the simple reuse of a class in a library, or via inheritance, whereby a new class may be created as an extension of an existing one behavior of inherited methods can be changed because of methods that are called within methods have to be tested per class . The object-oriented paradigm has numerous other powerful features including inheritance, data abstraction, and dynamic binding. These testing features not possible in traditional testing. If a fault in an inherited function is encountered only in the context of the derived class, then this fault cannot be detected without the selected testing technique forcing an invocation of this function in an object, which binds to this derived class. Our study [3] suggests that traditional testing techniques, such as functional testing, statement testing and branch testing, are not viable for detecting OO faults. To overcome these deficiencies, it is necessary to adopt an object-oriented testing technique that takes these features into account. However, the extent to which the cost and benefit we can balance by adopting an object oriented testing depends on how the program under test has been implemented.

Test case design methods for OO software are still evolving. However, an overall approach to OO test case design has been defined by Berard [8]:

1. Each test case should be uniquely identified and explicitly associated with the class to be tested.
2. The purpose of the test should be stated.
3. A list of testing steps should be developed for each test and should contain [8]:
   a. A list of specified states for the object that is to be tested.
   b. A list of messages and operations that will be exercised as a consequence of the test.
   c. A list of exceptions that may occur as the object is tested.
   d. A list of external conditions.
   e. Supplementary information that will aid in understanding or implementing the test.

Unlike conventional test case design, which is driven by an input-process-output view of software or the algorithmic detail of individual modules, object-oriented testing focuses on designing appropriate sequences of operations to exercise the states of a cl ass. Object-oriented Software is developed incrementally with iterative and recursive cycles of planning, analysis, design, implementation and testing .testing plays a special role here, since it is done after each increment [4].

3.1. Artifacts of Object Oriented Software

A. Attributes which plays important role makers of OO Software [5].

a) Encapsulation
A wrapping up of data and functions into a single unit is known as encapsulation. This restricts visibility of object states and also restricts 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 be inherited in the derived class too. A subclass can't be tested without its super classes. Abstract classes can't be tested at all
c) Polymorphism
Polymorphism is one of the crucial features of OOP. It simply means one name multiple forms. Because of polymorphism, all possible bindings have to be tested. All potential execution paths and potential errors have to 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 walkthrough, reviews should be conducted to ensure correctness, completeness and consistency

Object – Oriented programming is centered on concepts like Object, Class, Message, Interfaces, Inheritance, Polymorphism etc., Traditional testing techniques can be adopted in Object Oriented environment by using the following techniques:
– Function based
– Class testing
– Integration testing
– Fault-Based testing
– Scenario Based testing

A. Function Based Testing
Function based testing is just like conventional (Traditional) testing is based on product requirement and specification.

B. Class Testing:
Class testing is performed on the smallest testable unit in the encapsulated class. Each operation as part of a class hierarchy has to be tested because its class hierarchy defines its context of use. New methods, inherited methods and redefined methods within the class have to be tested. This testing is performed using the following approaches:

• Test each method (and constructor) within a class
• Test the state behavior (attributes) of the class between methods

Class testing is different from conventional testing in that Conventional testing focuses on input-process-output, whereas class testing focuses on each method. In addition to testing methods within a class (either glass box or black box). Test cases should be designed so that they are explicitly associated with the class and/or method to be tested. The purpose of the test should be clearly stated. Each test case should contain:

1. A list of messages and operations that will be exercised as a consequence of the test
2. A list of exceptions that may occur as the object is tested.
3. A list of external conditions for setup (i.e., changes in the environment external to the software that must exist in order to properly conduct the test)
4. Supplementary information that will aid in understanding or implementing the test

Some challenge in class testing [6].
1. Encapsulation:
   – Difficult to obtain a snapshot of a class without building extra methods which display the classes' state

2. Inheritance and polymorphism:
   – Each new context of use (subclass) requires re-testing because a method may be implemented differently (polymorphism).
   – Other unaltered methods within the subclass may use the redefined method and need to be tested

3. White box tests:
   – Basis path, condition, data flow and loop tests can all apply to individual methods, but don't test interactions between methods

Class level testing classified into following parts:

1. Random class testing
Identify methods applicable to a cl ass. Define constraints on their use – e.g. the class must always be initialized first. Identify a minimum test sequence – an operation sequence that defines the minimum life history of the class. Generate a variety of random (but valid) test sequences – this exercises more complex class instance life histories

2. Partitioned Based Testing
Reduces the number of test cases required to test a class in much the same way as equivalence partitioning for conventional software following type of partitioned based testing:
   – state-based partitioning:
   Tests designed in way so that operations that cause state changes are tested separately from those that do not

– attribute-based partitioning:
For each class attribute, operations are classified according to those that use the attribute, modify the attribute & do not use or modify the attribute.

– category-based partitioning:
Operations are categorized according to the function they perform:
i. Initialization.
ii. Computation
iii. Query
iv. Termination

C. Integration Testing:
OO does not have a hierarchical control structure so conventional top-down and bottom up integration tests have little meaning. Integration testing can be applied in three different incremental strategies:

- Thread-based testing, which integrates classes required to respond to one input or event.
- Use-based testing, which integrates classes required by one use case.
- Cluster testing, this integrates classes required to demonstrate one collaboration.

Test cases should be designed so that they are explicitly associated with the class and/or method to be tested. The purpose of the test should be clearly stated. Each test case should contain:

- A list of messages and operations that will be exercised as a consequence of the test
- A list of exceptions that may occur as the object is tested
- A list of external conditions for setup (i.e., changes in the environment external to the software that must exist in order to properly conduct the test)
- Supplementary information that will aid in understanding or implementing the test

D. Fault – Based Testing
Any product must conform to Customer requirements. Hence, testing should begin with the analysis model itself to uncover errors. Fault – Based testing is the method used to design tests that have a high probability finding probable errors of the software [7]. Fault – Based testing should begin with the analysis and design models. This type of testing can be based on the specification (user's manuals, etc.) or the code. It works best when based on both.

E. Scenario – Based Testing this new type of testing concentrates on what the customer does, not what the product does. It means capturing the tasks (use cases, if you will) the customer has to perform, then using them and their variants as tests. Of course, this design work is best done before you've implemented the product. It's really an offshoot of a car eful attempt at "requirements elicitation". These scenarios will also tend to flush out interaction bugs. They are more complex and more realistic than fault based tests often are. They tend to exercise multiple subsystems in a s ingle test, exactly because that's what users do. The tests won't find everything, but they will at least cover the higher visibility interaction bugs [7].

4. Objective Of Research

In this research work consists of:

- Design object oriented of testing architecture Template at class diagram.
- Using this architecture we r epresents different operation of each testing technique and associated different attribute, using some o peration of testing technique with others testing operation (has set of operations it is capable of performing to change its attribute values which may cause changes to attribute values of other objects) .
- Here we try providing framework for comprehensive object –oriented testing tool.

In figure1 object oriented testing divide into three parts based on their functionality.
First category consists of functional testing, class testing and its derived classes in this category directly based on requirement and specification of software products

1. Input the functional specification for function level testing any testing tool.
2. According functional specification constructs class level testing.
3. Class level testing divide into two parts partitioning class testing and random testing.

Partitioning based testing & random testing are derived from class level testing its use some properties of class testing.

Second category Integration based testing its further divide into three parts thread, cluster and used based testing.

1. 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.
2. Use-based testing, begins the construction of the system by testing those classes (called independent classes) that use very few (if any) of 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.

3. Cluster testing is one step in the integration testing of OO software. Here, a cluster of collaborating classes (determined by examining the C RC and object-relationship model) is exercised by designing test cases that attempt to uncover errors in the collaborations.

Third parts consist of fault based testing and scenario based testing.
1. The object 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 plausible faults (i.e., aspects of the implementation of the system that may result in defects). To determine whether these faults exist, test cases are designed to exercise the design or code.
2. Fault-based testing misses two main types of errors: (1) incorrect specifications and (2) interactions among subsystems. When errors associated with incorrect specification occur, the product doesn't 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. But to accomplish this, test cases must be more complex and more realistic than fault-based tests. Scenario-based testing tends to exercise multiple subsystems in a single test.

5. Conclusion

The maturation of testing techniques has been fruitful, but not adequate. Pressure to produce higher-quality software at lower cost is increasing and existing techniques used in practice are not sufficient for this purpose. Fundamental research that addresses the challenging problems, development of methods and tools, and empirical studies should be carried out so that we can expect significant improvement in the way we test software. Researchers should demonstrate the effectiveness of many existing techniques for large industrial software, thus facilitating transfer of these techniques to practice. The successful use of these techniques in industrial software development will validate the results of the research and drive future research. The pervasive use of software and the increased cost of validating it will motivate the creation of partnerships between industry and researchers to develop new techniques and facilitate their transfer to practice. Development of efficient testing techniques and tools that will assist in the creation of high-quality software will become one of the most important research areas in the near future.

In this research work first establish a total set of requirement specification for a comprehensive software testing tool. In Object Oriented environment, these requirements will address various testing methods and strategies object oriented development scenarios. This work will propose architectural designs object oriented paradigms which will satisfy the established requirements specifications .These designs can be further translated into practical industrial tools.

Future Work

Also, this study will propose set of metrics which will be relevant to do m easurements on the proposed architectures. These measurements will be used to draw inferences for understanding behavior of the metrics in relation to the proposed architectures for improving the designs for optimizing their quality

6. References:

[1]. Edward Miller and William E. Howden. Tutorial: Software Testing & Validation Techniques. IEEE Computer Society Press, second edition, 1981. [23] John D. Musa. A theory of software reliability and its applications. IEEE Transactions.

[2]. A. J. J. Marciniak, “Encyclopedia of software engineering”, Volume 2, New York, NY: Wiley, 1994, pp.1327-1358.

[3]. G. M. Kao, M. H. Tang, and M. H. Chen. Investigating test effectiveness on object oriented software - a case study. In Proceedings of Twelfth Annual International Software Quality Week, 1999.

[4]. Jilles van Gurp,Oject Oriented Testing Reports,software verification and validation,DAD404, IDE, University of karlskrona/Ronneby,1998

[5]. G.Suganya, S .Neduncheliyan, A Study of Object Oriented Testing Techniques: Survey and Challenges.

[6]. Based on n otes from James Gain (jgain@cs.uct.ac.za) Larman, chapter 21and notes on Larman from George Blank of NJIT plus Glenn Blank’s elaborations and expansions.

[7]. Roger S.Pressman “Software Engineering –A Practitioner’s Approach” McGraw Hill International Edition.

[8]. Berard, E.V., Essays on Object-Oriented Software Engineering, vol. 1, Addison-Wesley, 1993

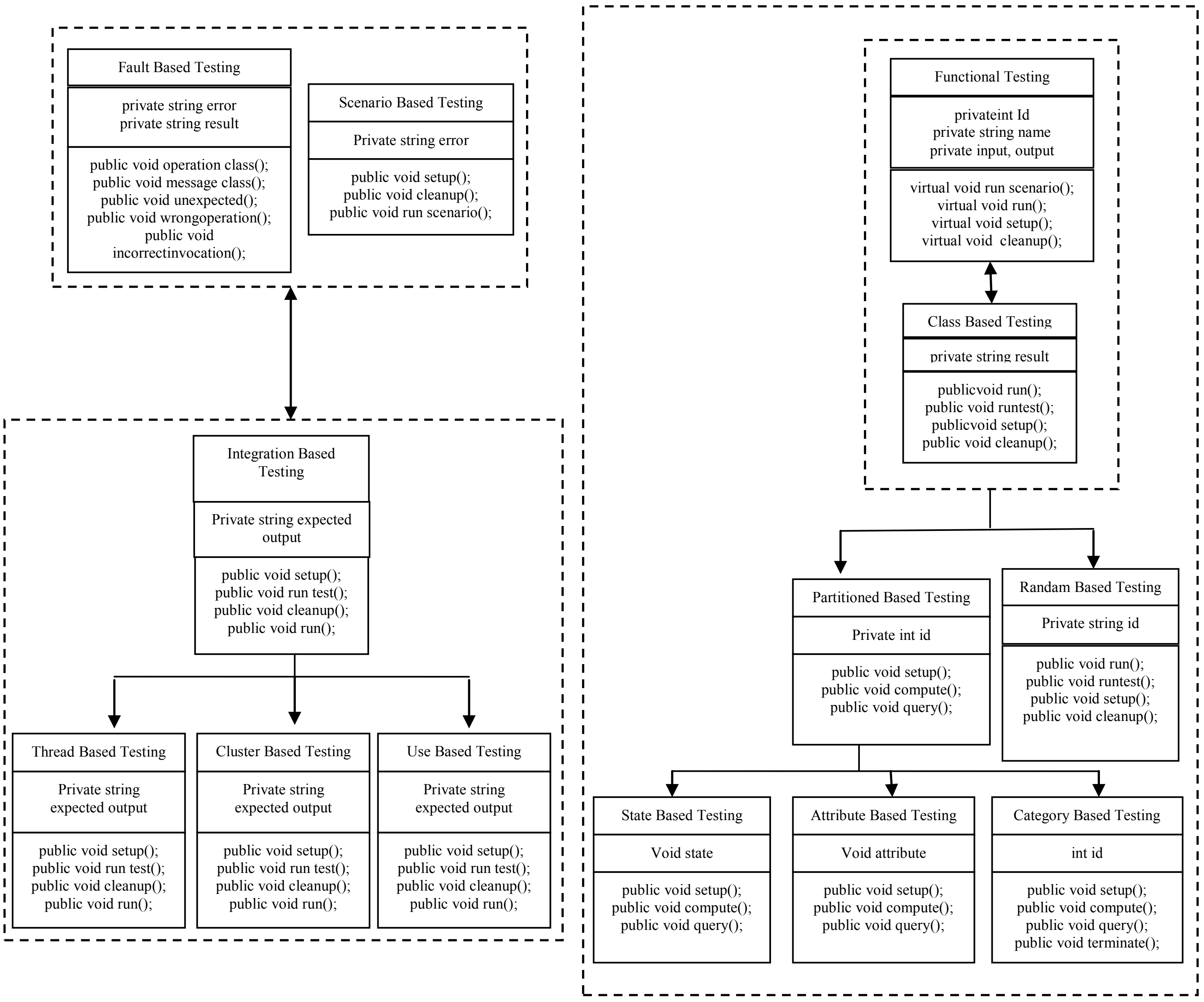

FIGURE 1: OBJECT ORIENTED ARCHITECTURE TOOL