SOFTWARE TESTING–A STUDY

Avinash H.Hedaoo[1], Mrs. Abha Khandelwal[2]
[1]Deptt of Comp. Sci, Dr. Ambedkar College, India
[2]Deptt of Comp. Sci, Hislop College, India

ABSTRACT

Software testing provides a means to reduce errors, cut maintenance and overall software costs. Numerous software development and testing methodologies, tools, and techniques have emerged over the last few decades promising to enhance software quality. While it can be argued that there has been some improvement it is apparent that many of the techniques and tools are isolated to a specific lifecycle phase or functional area. One of the major problems within software testing area is how to get a suitable set of cases to test a software system. This set should assure maximum effectiveness with the least possible number of test cases. There are now numerous testing techniques available for generating test cases.

KEYWORDS: SOFTWARE TESTING, LEVEL OF TESTING, TESTING TECHNIQUE, TESTING PROCESS, TEST MANAGEMENT

1. INTRODUCTION

Software testing comprises a major part of software development lifecycle. In past those software not tested resulted in social problems and financial losses. By one estimate defective software causes approximately US $ 50 B losses to USA each year [1]. This estimate suggests industry-wide deficiency in testing. According to Bertolino [2] testing is a general validation approach in industry, but it is still largely ad hoc, expensive, and its effectiveness is not predictable. Exhaustive testing is not possible as we face lack of time and resources. As there are limited resources available for testing we should select effective and efficient testing techniques. The lack of sufficient information about effectiveness, efficiency and cost of testing techniques makes difficult selection of a testing technique. Assessing effectiveness and efficiency of a testing technique is not easy as there are various operations involved in testing which depends on the subject that applies it, the programming language, software under test, the type of faults etc. Some advances have been made in evaluating effectiveness, efficiency of testing techniques but there is still a long way to go as results are very inconclusive.

2. THE HISTORY OF TESTING TECHNIQUES

2.1 CONCEPT EVOLUTION

Software has been tested as early as software has been written. The concept of testing itself evolved with time. The evolution of definition and targets of software testing has directed the research on testing techniques. The concept evolution of testing using the testing process model proposed by Gelperin and Hetzel [3] is as follows :

Phase I. Before 1956: The Debugging-Oriented Period – Testing was not separated from debugging

Phase II. 1957~78: The Demonstration-Oriented Period – Testing to make sure that the software satisfies its specification

Phase III. 1979~82: The Destruction-Oriented Period – Testing to detect implementation faults

Phase IV. 1983~87: The Evaluation-Oriented Period – Testing to detect faults in requirements and design as well as in implementation

Phase V. Since 1988: The Prevention-Oriented Period – Testing to prevent faults in requirements, design, and implementation

2.2 DEFINITION OF TESTING

The definition of testing according to the ANSI/IEEE 1059 standard is that testing is the process of analysing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

3. SOFTWARE TESTING TECHNIQUES

Testing technique is a method or approach that systematically describes how set of test cases should be created (with what intention and goals) keeping into consideration possible rules for applying test cases. Testing techniques aids in limiting the number of test cases that can be created [4].

Software testing techniques can be classified into two main categories based on the criteria whether the technique requires actual execution or not [5]:

3.1 Static Testing

3.2 Dynamic Testing

3.1 STATIC TESTING TECHNIQUES

Static techniques are concerned with the analysis and checking of system representations such as the requirements documents, design diagrams and the program source code, either manually or automatically, without actually executing the code [6].

Static techniques can be grouped under two categories:

3.1.1 Reviews

3.1.2 Analysis

3.1.1 REVIEWS

A software review is "A process or meeting during which a software product is examined by a project personnel, managers, users, customers, user representatives, or other interested parties for comment or approval"[7].

According to IEEE 1028 Phases of formal review [8] are : Planning, Kick-off, Individual Preparation, Review Meeting Rework and Follow-up. Roles and responsibilities are : **Manager ,Moderator, Reviewers and Scribe (Recorder)**

Reviews can be of following types :

3.1.1.1 INFORMAL REVIEW

A review done by peers in an informal fashion without any documented findings or process. An informal review involves two or more people looking through a document or codes that one or the other of them has written. The purpose is still to detect defects, but there are usually no check-lists used and the result does not need to be documented [9].

3.1.1.2 WALKTHROUGH

In software engineering, a walkthrough or walk-through is a form of software peer review "in which a designer or programmer leads members of the development team and other interested parties through a software product, and the participants ask questions and make comments about possible errors, violation of development standards, and other

problems"[10]. The general goals of a walkthrough are : to present the document to stakeholders; knowledge transfer; to establish a common understanding and to establish consensus.

3.1.1.3 TECHNICAL REVIEW

A software technical review is a form of peer review in which "a team of qualified personnel examines the suitability of the software product for its intended use and identifies discrepancies from specifications and standards. Technical reviews may also provide recommendations of alternatives and examination of various alternatives".

3.1.1.4 INSPECTION

Inspection in software engineering of any work product is looking for defects using a well defined process, it is also called as peer review and done by trained individuals. An inspection might also be referred to as a Fagan inspection after Michael Fagan, the creator of a very popular software inspection process.

3.1.1.5 MANAGEMENT REVIEW

A systematic evaluation of a software acquisition, supply, development, operation, or maintenance process performed by or on behalf of management that monitors progress, determines the status of plans and schedules, confirms requirements and their system allocation, or evaluates the effectiveness of management approaches used to achieve fitness for purpose [11].

3.1.1.6 AUDIT

Audit is the most formal static testing technique. They are conducted by personnel external to a project to evaluate compliance with specifications standards contractual agreements or other criteria [12]. There are two types of Audit internal and external.

3.1.2 ANALYSIS

Static analysis is performed on requirements, design or code without actually executing the software artifact being examined. It is ideally performed before the types of formal review and unrelated to dynamic properties of the requirements, design and code, such as test coverage. The goal of static analysis is to find defects, whether or not they may cause failures. As with reviews, static analysis finds defects rather than failures. Static Analysis can

reduce defects by up to a factor of six [12]. Static Analysis is done using approaches like Coding Standards, Code Metric and Code Structure.

3.2 DYNAMIC TESTING TECHNIQUES

Dynamic execution based techniques focus on the range of ways that are used to ascertain software quality and validate the software through actual executions of the software under test. We test the software with real or simulated inputs, both normal and abnormal, under controlled and expected conditions to check how a software system reacts to various input test data. It is essential to test the software in controlled and expected conditions as a complex, non deterministic system might react with different behaviors to a same input, depending on the system state. Dynamic testing techniques are generally divided into the two broad categories block box testing and white box testing [13][14].

3.2.1 BLACK BOX TESTING

Black box testing is based on the requirements specifications and there is no need to examining the code. This is purely done based on customers view point only tester knows the set of inputs and predictable outputs. Black box testing is done on the completely finished product [15] [16]. Black box testing techniques are Equivalence Class Partitioning, Boundary Value Analysis, Decision Tables, State Transition Diagrams, Orthogonal Arrays and All Pairs Technique

3.2.2 WHITE BOX TESTING

White box testing mainly focus on internal logic and structure of the code. White-box is done when the programmer has full knowledge of the program structure. With this technique it is possible to test every branch and decision in the program [17] [18] [9]. White box testing techniques are Static white box testing and Structural White box testing. Static white box testing further divided into Desk checking, Code walkthrough and Formal Inspections. Structural White box testing further divided into Control flow/ Coverage testing, Basic path testing, Loop testing and Data flow testing

4 TESTING LEVELS

Whichever the process adopted, we can at least distinguish in principle between unit, integration and system test [19], [20] that are distinguished by the test target without implying a specific process model. Other test levels are classified by the testing objective.

4.1 UNIT TESTING

Unit testing or component testing is used to ensure that if the unit is working as per its functional specification and/or that its design matches the intended design or not [19], [20]. Unit tests can also be applied to check interfaces, local data structure or boundary conditions[21]. These types of tests are usually written by developers as they work on code (white-box style). One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing is used to ensure that units are working independently of each other or not, unit testing alone cannot verify the functionality of a piece of software. Unit testing is used to make development and QA process efficient and to increase quality of the software by eliminating errors while software is in construction before the code is promoted to QA. Depending on the organization's expectations for software development, unit testing might include static code analysis, data flow analysis metrics analysis, peer code reviews, code coverage analysis and other software verification practices[22]. The unit testing helps by three ways. First, attention is focused initially on smaller units of the program. Second, when an error is found, it is known to exist in a particular module. Finally, multiple modules can be tested simultaneously[18]. Tools for unit testing are Debug, Re-structure, Code Analyzers, Path/statement coverage tools.

4.2 INTEGRATION TEST

In Integration testing (sometimes called integration and testing, abbreviated I&T) individual software modules which have been unit tested are combined as larger aggregates and tested as a group as per the integration test plan . Integration testing delivers as its output the integrated system ready for system testing. It occurs after unit testing and before validation testing[23].

In this testing major design items are tested to ensure their functional, performance, and reliability status against requirements. Black box testing, success and error cases being simulated via appropriate parameter and data inputs are used to exercise these "design items", i.e. assemblages (or groups of units) through their interfaces. Simulated usage of shared data areas and inter-process communication is tested and individual subsystems are exercised

through their input interface. The overall idea is a "building block" approach, in which verified assemblages are added to a verified base which is then used to support the integration testing of further assemblages.

Some different types of integration testing are big bang, top-down, and bottom-up. Other Integration Patterns[24] are: Collaboration Integration, Backbone Integration, Layer Integration, Client/Server Integration, Distributed Services Integration and High-frequency Integration.

4.3 SYSTEM TEST

In system testing software under test is implemented in its actual hardware environment and tested to verify that the system behaves according to the user requirements. Goals of system testing can be [25]:

- to discover system level failures which were not detected during unit or integration testing;
- to make sure that developed product correctly implements the required capabilities;
- to gather data useful for deciding the release of the product.

System testing should therefore ensure that each system function works as expected, any failures are exposed and analyzed, and additionally that interfaces for export and import routines behave as required. Generally system testing includes testing for performance, security, reliability, stress testing and recovery [19]. In particular, test and data collected applying system testing can be used for defining an operational profile necessary to support a statistical analysis of system reliability [26].

A further test level, called **Acceptance Test,** is often added to the above subdivision. It is mainly focuses on the usability requirements.

4.4 REGRESSION TEST

Properly speaking, regression test is not a separate level of testing, but may refer to the retesting of a unit, a combination of components or a whole system (see Fig. 1 below) after modification, in order to ascertain that the change has not introduced new faults [19].

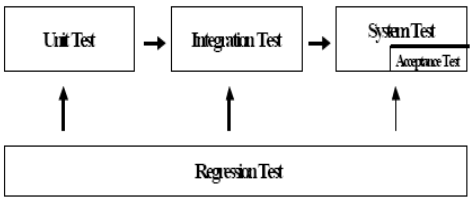

Figure 1. Logical Schema of Software Testing Levels

Selective regression test techniques [27] help in selecting a (minimized) subset of the existing test cases by examining the modifications (for instance at code level, using control flow and data flow analysis).

5 TESTING TYPES

Following are the types of testing:

5.1 INSTALLATION TESTING

An installation test assures that the system is installed correctly and working at actual customer's hardware.

5.2 COMPATIBILITY TESTING

Compatibility testing is one of the test types performed by testing team. Compatibility testing checks if the software can be run on different hardware, operating system, bandwidth, databases, web servers, application servers, hardware peripherals, emulators, different configuration, processor, different browsers and different versions of the browsers etc.,

5.3 SMOKE AND SANITY TESTING

Initial effort to determine if a new software version is performing well enough to accept it for a major testing effort.

5.4 REGRESSION TESTING

As discussed in section 4.4.

5.5 ACCEPTANCE TESTING

Acceptance testing can mean one of two things:

i. A smoke test is used as an acceptance test prior to introducing a new build to the main testing process, i.e. before integration or regression.

ii. Acceptance testing performed by the customer, often in their lab environment on their own hardware, is known as user acceptance testing (UAT). Acceptance testing may be performed as part of the hand-off process between any two phases of development.

5.6 ALPHA TESTING

Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site. [28]

5.7 BETA TESTING

Beta testing comes after alpha testing and can be considered a form of external user acceptance testing.

5.8 FUNCTIONAL VS NON-FUNCTIONAL TESTING

Functional testing refers to activities that verify a specific action or function of the code. 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 other performance, behavior under certain constraints, or security.

5.9 DESTRUCTIVE TESTING

Destructive testing attempts to cause the software or a sub-system to fail. It verifies that the software functions properly even when it receives invalid or unexpected inputs, thereby establishing the robustness of input validation and error-management routines.

5.10 SOFTWARE PERFORMANCE TESTING

Performance testing is generally executed to determine how a system or sub-system performs in terms of responsiveness and stability under a particular workload. It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage.

There is little agreement on what the specific goals of performance testing are. The terms load testing, performance testing, scalability testing, and volume testing, are often used interchangeably.

5.11 USABILITY TESTING

Usability testing is needed to check if the user interface is easy to use and understand. It is concerned mainly with the use of the application.

5.12 ACCESSIBILITY TESTING

Accessibility testing may include compliance with standards such as: Americans with Disabilities Act of 1990, Section 508 Amendment to the Rehabilitation Act of 1973 and Web Accessibility Initiative (WAI) of the World Wide Web Consortium (W3C)

5.13 SECURITY TESTING

Security testing is essential for software that processes confidential data to prevent system intrusion by hackers.

6. STRATEGIES FOR TEST CASE SELECTION

Strategies are used to increase effectiveness of testing in terms of thoroughness on one side and reducing times and costs on the other. It is very important that how the test cases are selected, as there are limited test resources.

6.1 SELECTION CRITERIA BASED ON CODE

During the late 70's and the 80's the trend was dominated by Code-based testing, also called “structural testing”, or “white box” testing in software testing research. For designing test cases for white-box testing an internal view of the system, as well as programming skills are used . The tester selects inputs to exercise paths through the code and find out the appropriate outputs.

White-box testing is generally done at the unit level, it can be applied at the unit, integration and system levels of the software testing process. It can check paths within a unit, paths between units during integration, and between subsystems during a system–level test. This method of test design uncover many errors or problems, but unimplemented parts of the specification or missing requirements might left undetected. Techniques used in white-box testing include: API testing, Code coverage, Fault injection, Mutation testing methods and Static testing methods .

6.2 SELECTION CRITERIA BASED ON SPECIFICATIONS

In specification-based testing, documentation relative to program specifications is generally used to derive reference model RM[29].

In Specification-based testing functionality of software is checked according to the applicable requirements.[30] In this testing detailed test cases should be provided to the tester, who then can simply apply the test cases to check that for a given input, whether the output value (or behavior), "is" or "is not" the same as the expected value specified in the test case. Test cases are designed based on specifications and requirements, i.e., what the application is supposed to do. To derive test cases it uses external descriptions of the software, including specifications, requirements, and designs. These tests can be functional or non-functional, though usually functional.

Using Specification-based testing correct functionality may be assured, but it is not sufficient to watch against complex or high-risk situations.[31]

This method of test can be useful in all levels of software testing: unit, integration, system and acceptance. It typically uses in most of the testing at higher levels, but can also dominate unit testing as well.

6.3 OTHER CRITERIA

Some other important strategies for test selection are briefly overviewed as follows :

6.3.1 BASED ON TESTER'S INTUITION AND EXPERIENCE

The testing technique based on tester's intuition and experience is ad-hoc testing techniques it is most widely practiced technique [32] in which tests are developed based on the tester's skill, intuition, and experience with similar programs. The special tests not easily captured by formalized techniques might be identified by Ad hoc testing. The testing technique in which test cases are developed during testing not before, is called Exploratory testing. A subset of exploratory testing is so-called guerilla testing. In this testing experienced tester tests limited section of program exhaustively for short period of time[33].

6.3.2 MUTATION BASED TEST CASE GENERATION

In testing for reliability evaluation, Mutation adequacy is the main fault-based test adequacy criterion. The competent programmer hypothesis and the coupling effect are two assumptions upon which the idea of a mutation adequate test suite is based [34]. The

competent programmer hypothesis assumption states that competent programmers develop code that compile and very nearly correct as per their specification. The coupling effect assumes that tests that detect simple faults are likely to also detect complex faults [34].

6.3.3 BASED ON OPERATIONAL USAGE

In testing for reliability evaluation, the test environment must reproduce the operational environment of the software as closely as possible (operational profile). It is done for assessing future reliability of the software when in actual use from the observed test results. To do this, inputs are assigned a probability distribution, or profile, as per their occurrence in actual operation.[35]

7. TEST DESIGN

Test design is a important phase of software testing, in which test suites for the objectives and the features to be tested are defined. Testing levels are planned and kind of approach for each level and for each feature to be tested is decided . Stopping criteria for testing also decided in test design. It is also decided that which part of the software should be tested more keeping in view time or budget constraints. [35].

8. TEST EXECUTION

There are various types of difficulties in executing the Test cases specified in test design. The various tasks involved in launching the test and deciding the outcome of the test are discussed below. Tools for testing activities automation also given below.

8.1 LAUNCHING THE TESTS

Test cases can be executed manually or automatically.

8.2 TEST ORACLES

The “test oracle” is a concept describing a method that is used to recognize correct and incorrect test output during software testing. When a test case is inputted to the system under test output comes, tester compares this output with the output specified in test oracle to ensure that the output is expected output or not. The term was first used and defined in William Howden's Introduction to the Theory of Testing. Additional work on different kinds of oracles was explored by Elaine Weyuker. [36].

8.3 TEST TOOLS

Program testing and fault detection can be done using testing tools and debuggers. Testing/debug tools contains features such as: Program monitors - performs full or partial monitoring of program code including: Instruction set simulator- performs complete instruction level monitoring and trace facilities, Program animation- performs step-by-step execution and conditional breakpoint at source level or in machine code and Code coverage reports, Formatted dump or symbolic debugging, tools performs inspection of program variables on error or at chosen points. Automated functional GUI testing tools are used to repeat system-level tests through the GUI Benchmarks, allowing run-time performance comparisons to be made. Performance analysis (or profiling tools) that can help to highlight hot spots and resource usage Some of these features may be incorporated into an Integrated Development Environment (IDE)[7]. In the following of this section we present a list of typologies of most commonly used test tools:

Test harness (drivers, stubs): Test Harness contains two main parts: the test execution engine and the test script repository. Test harnesses is used to automate tests. It helps in integration testing. In integration testing test stubs are typically module of the software under development and are replaced by working module as the application is developed (top-down design), test harnesses are external to the application being tested and simulate services or functionality not available in a test environment. [37].

Test generators: It is used to generate tests.

Capture/Replay: this tool can automatically re-executes, or replays, previously run tests, on the basis recorded inputs and outputs (e.g., screens).

Oracle/file comparators/assertion checking: By using these types of tool we can decide whether a test is pass or fail;

Coverage analyzer/Instrumenter: A coverage analyzer is used to assess which and how many components of the program flowgraph have been exercised. The program instrumenters insert probes into the code which helps in analysis.

Tracers: It shows the history of execution of a program;

Reliability evaluation tools: Using this tool we can analyse test results graphically. Using these test results we can assess reliability related measures according to selected models.[35]

9. TEST DOCUMENTATION

A summary of the ANSI/IEEE Standard 829-1983 describes a test plan as: "A document depicting the scope, approach, resources, and schedule of intended testing processes. It recognizes test items, the units to be tested, the testing tasks, who will do each task, and any proactive strategy for risks." This standard describes the following test plan outline: Test Plan Identifier, Introduction, Test Items, Features to be Tested, Features Not to Be Tested, Approach, Item Pass/Fail Criteria, Suspension Criteria and Resumption Requirements, Test Deliverables, Testing Tasks, Environmental Needs, Responsibilities, Staffing and Training Needs, Schedule, Risks and Contingencies and Approvals[38].

10. TEST MANAGEMENT

Test management manages the computer software testing process such as organizing test assets and artifacts such as test requirements, test plans, test cases, test scripts and test results to enable easy accessibility and reusability . Test artifact and resource organization is necessary part of test management.

11. TEST MEASUREMENTS

In the field of software engineering test measurement helps in generating description of important processes and products quantitatively, and accordingly controlling software behavior and results. Also nature and impact of proposed changes can be understood by using measurement as baseline. In addition, managers and developers use measurement to monitor effects of tasks and changes on all facets of development. In this way whether or not final product meets plans can be checked as early as possible and accordingly actions can be taken [39]. In case of testing phase, measurement can be used for evaluation of the program under test, or the selected test set, or even for monitoring the testing process itself [40].

12. CONCLUSIONS

In this paper we have presented a detailed study of software testing. The approaches overviewed include more traditional techniques, e.g., code-based criteria, as well as more modern ones, such as model checking. We have tried to contribute to the best of our knowledge by putting into all the possible details about software testing discipline, through this paper we have tried to demonstrated that testing is a very complex activity and should be given importance in software development. Through this paper, we have tried to attract interest from academy and industry. However, what we can and must pursue is to transform

testing from "trial-and-error" to a systematic, cost-effective and predictable engineering discipline.

REFERENCES

1. NIST-Final Report "*The Economic Impacts of Inadequate Infrastructure for Software Testing", Table 8-1, National Institute of Standards and Technology, May 2002.*
2. Bertolino, A. (2007). , "*Software testing research: Achievements, challenges, dreams.", In FOSE '07: 2007 Future of Software Engineering, pages 85–103, Washington, DC, USA. IEEE Computer Society*.
3. D. Gelperin and B. Hetzel*, "The Growth of Software Testing", Communications of the ACM, Volume 31 Issue 6, June 1988, pp. 687-695*
4. Eldh 2011
5. Roper 1995
6. Sommerville, I, *"Software Engineering", Pearson, 2008, 864 pages, ISBN 978-81-317-2461-3*
7. IEEE Std . 1028-1997, *"IEEE Standard for Software Reviews", clause 3.5*
8. Fagan, Michael E: *"Design and Code Inspections to Reduce Errors in Program Development", IBM Systems Journal, Vol. 15, No. 3, 1976; "Inspecting Software Designs and Code", Datamation, October 1977; "Advances In Software Inspections", IEEE Transactions in Software Engineering, Vol. 12, No. 7, July 1986*
9. Weinberg, Gerald M., *"The psychology of computer programming", Dorset House Pub.,1998, 292 pages*
10. IEEE Std. 1028-1997, *IEEE Standard for Software Reviews, clause 3.8*
11. IEEE Std 1028-1997, *"IEEE Standard for Software Reviews, IEEE Computer Society, ISBN 1-55937-987-1*
12. Software Audit Review, *http://en.wikipedia.org/wiki/Software_audit_review, Accessed on 12 Oct 2013*
13. Sommerville, 2007
14. Beizer , 1995
15. P. Mitra, S. Chatterjee, and N. Ali, *"Graphical analysis of MC/DC using automated software testing," in Electronics Computer Technology (ICECT), 2011 3rd International Conference on, 2011, vol. 3, pp. 145 –149.*
16. T. Murnane and K. Reed, *"On the effectiveness of mutation analysis as a black box testing technique," in Software Engineering Conference, 2001. Proceedings. 2001 Australian, 2001, pp. 12 –20*

17. P. Mitra, S. Chatterjee, and N. Ali, *"Graphical analysis of MC/DC using automated software testing," in Electronics Computer Technology (ICECT), 2011 3rd International Conference on, 2011, vol. 3, pp. 145 –149.*

18. Glenford J. Myers ; Revised and updated by Tom Badgett and Todd *The Art of Software Testing, Second Edition Thomas, with Corey Sandler.—2nd ed. p. cm. ISBN 0-471-46912-2*

19. S.L. Pfleeger, *Software Engineering Theory and Practice, Prentice Hall, 2001.*

20. B. Beizer, Software Testing Techniques 2nd Edition, International 13 Thomson Computer Press, 1990.

21. IEEE Standard for *Software Unit Testing IEEE Std. 1008-1987 (R1993).*

22. Wikipedia, *Software testing*

23. *M*artyn A Ould & Charles Unwin (ed), *Testing in Software Development, BCS (1986), p71. Accessed 31 Oct 2014*

24. Binder, Robert V.: *Testing Object-Oriented Systems: Models, Patterns, and Tools. Addison Wesley 1999. ISBN 0-201-80938-9*

25. Sevinç Ozkara, Cryptographic Techniques- Essential Applications In Network Administration, Scholedge International Journal Of Multidisciplinary & Allied Studies, Vol.1, Nov. Issue; www.scholedge.org

26. R.V. Binder Testing Object-Oriented Systems - Models, Patterns, and Tools, Addison-Wesley, 1999.

27. M.R Lyu, eds., *Handbook of Software Reliability Engineering*, McGraw-Hill, 1996.

28. G. Rothermel. and M.J. Harrold, *"Analyzing Regression Test Selection Techniques", IEEE Transactions on Software Engineering, vol. 22, no. 8, pp. 529 – 551, 1996.*

29. Van Veenendaal, Erik. *"Standard glossary of terms used in Software Testing". Retrieved 4 January 2013*

30. P. C Jorgensen, *Software Testing a Craftsman's Approach. CRC Press, 1995.*

31. Laycock, G. T. (1993). *The Theory and Practice of Specification Based Software Testing (PostScript). Dept of Computer Science, Sheffield University, UK. Retrieved 2008-02-13.*

32. Bach, James (June 1999). "Risk and Requirements-Based Testing" (PDF). Computer 32 (6): 113–114. Retrieved 2008-8-19.

33. Ram Chillarege ,*"Software Testing Best Practices" , IBM Technical Report RC 21457 Log 96856 April 26, 1999.*

34. Manfred Ratzmann and Clinton De Young, Galelio *Computing Software Testing and Internationalization 2003*. DeMillo et al., 1988

35. Antonia Bertolino, Eda Marchetti, *A Brief Essay on Software Testing*

36. Http://en.wikipedia.org/wiki/Oracle_(software_testing)

37. Http://en.wikipedia.org/wiki/Test_harness

38. Www.sqatester.com/documentation/.../IEEEStandardTestPlans.doc *IEEE Standard for Software Test Documentation (ANSI/IEEE Standard 829-1983)*

39. N.E. Fenton, and S.L Pfleeger *Software Metrics - A Rigorous and Practical Approach". Second ed. London: International Thomson Computer Press, 1997*

40. 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*