International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

An Innovative Approach to Study Various Software Testing Techniques and Strategies: Review Paper

Mr. V. Kiran Kumar, Dr. P. Vidya Sagar

Submitted: 15/05/2024 **Revised:** 26/06/2024 **Accepted:** 06/07/2024

Abstract— There are a few techniques for programmed experiment age has been proposed previously. In any case, most of these strategies are primary trying procedures that require the comprehension of the inside working of the program. Software testing is the process to uncover requirement, design and coding errors in the program. It is used to identify the correctness, completeness, security and quality of software products against a specification. Software testing is the process used to measure the quality of developed computer software There is less useful inclusion of all testing methods together. In this paper we conduct a literature review writing concentrate on all testing procedures together that are connected with both Highly contrasting box testing strategies. In this paper, we have described and compared the two most important and commonly used software testing techniques for detecting errors, which are: Black Box Testing and White Box Testing. A recommended procedure for model validation is presented and model accreditation is briefly discussed.

Index Terms—: testing techniques, white box testing, black box, grey box testing, literature review, verification, validation, simulation theory, etc.

I Introduction

Software testing is a method for identifying system flaws. It helps us find and fix system flaws, errors, faults, and failures. Since the idea of software development first emerged, numerous methods and strategies have been developed. The purpose of software testing is to maximize software quality. In this paper, we discuss the most common testing methods and approaches. What they can be used for and how to use them The following describes how they function and how they differ from one another. Techniques: Testing with a black box, white box, and grey box Strategies: Acceptance Testing, System Testing, and Unit Testing.[1]

According to Naik & Tripathy (2013), software testing is both a verification procedure for determining software quality and a method for achieving that quality. a crucial part of software development. It has been around as long as software development. The method by which a software system is tested determines its quality. Testers and organizations recommend allocating 40-50% of their time and funds to testing. The system must be properly tested in order to attain a high level of reliability, maintainability, availability, security, survivability, portability, capability, efficiency, and integrity.[4]

We can be certain that the system will function as intended thanks to software testing. Modern software must be accurate and provide all required functionality for critical applications. The organization, end user, developer, and tester all benefit greatly from testing. There are numerous testing methods and strategies for software testing. which our paper discusses. [5]

The term "testing" refers to the process of determining whether or not a given system satisfies the requirements it was designed to meet. It primarily consists of a validation and verification process to determine whether the developed system meets the user's requirements. As a result, the outcome of this activity differs from what was anticipated. Finding bugs, errors, or missing requirements in the developed system or software is known as software testing. Therefore, this is an investigation that provides the stakeholders with precise information regarding the product's quality .[3]

Software testing is also a risk-based activity that can be considered. During the testing process, it is essential for software testers to understand how to

Department of Computer Science & Engineering, Koneru Lakshmaiah Education Foundation Vaddeswaram,
Andhra Pradesh, India,
India E-mail: kirankumarvanapalli@gmail.com
Department of Computer Science & Engineering, Koneru Lakshmaiah Education Foundation,
Vaddeswaram, A.P, India E-mail: pvsagar20@gmail.com.

break down a large number of tests into manageable sets and make educated decisions regarding which risks should be tested and which should not [1].

There is a correlation between the testing cost and errors in Figure 1. Figure 1 makes it abundantly clear that testing of both functional and nonfunctional types significantly increases costs. The process of deciding what to test or how many to test can result in many bugs being missed. The objective of effective testing is to conduct the maximum number of tests possible to minimize additional testing effort [1].

Software testing is an important part of software quality assurance, as **shown in Figure 1**. Life-critical software testing, such as flight control testing, can illustrate the significance of testing by illustrating the risk of schedule delays, cost overruns, or even cancellation [2, and more about this [3][4].

There are various levels and steps to the testing, and the test taker varies from level to level. Unit testing, integration testing, and system testing are the three fundamental stages of software testing. The software developer or the quality assurance engineer, also known as a software tester, test each of these steps [5]. The Software Development Lifecycle (SDLC) includes all of the above-mentioned testing steps. The development of software must be broken down into a set of modules, with a different team or individual responsible for each module. The process of testing each module or unit to see if it performs as expected is referred to as "Unit Testing" after it has been completed by the developer. Integration Testing is the second stage of the SDLC's testing process. After the modules of a single software system have been developed independently, they are integrated together, and once the integration is complete, it is common for errors to occur during the build. System Testing, or testing the software as a whole from every angle, is the final testing step in the software development lifecycle (SDLC). Software testing also ensures that the integrated units do not disrupt or interfere with the programming of any other module. However, testing a large or extremely complex system can take a very long time because testing each combination becomes more difficult as the number of components in the application increases**.** and scenario, indicating a pressing need for a more effective software testing procedure for premium optimization [6].

From Test Planning to the analysis of Test Results, the testing cycle mostly consists of several phases. The plan for all of the test activities that will be carried out throughout the testing process is the primary focus of the first phase of test planning. The development of the test cases that will be utilized during the testing process takes place during the second phase of the testing life cycle, known as Test Development. The next phase of the Testing cycle is Test Execution, which includes executing the test cases. The relevant bugs are reported in the Test Reporting phase, which is the next phase. Test Result Analysis is the final stage of the testing process. Here, the software or system developer conducts a defect analysis. This step can also be handled with the client because it will help them better understand what to ignore and what to fix, improve, or just change [7].

II. Existing Testing Methods

The creation of test cases is the first step in the Testing procedure. To ensure efficient and accurate testing, a variety of testing methods are used to create the test cases.

Black box testing, white box testing, and grey box testing are the most common testing methods [8].

White Box testing is a method of testing that tests not only the software's functionality but also the application's internal structure, making it very effective.

It is necessary to have programming skills in order to design the test cases in order to carry out white box testing. Clear box and glass box testing are other names for white box testing. Unit, integration, and system testing are all examples of this kind of testing. The purpose of this kind of testing—also known as security testing—is to ascertain whether information systems safeguard data and continue to function as intended. Because this type of testing makes use of the software's internal logical arrangement, it can test all of a module's independent paths, exercise every logical decision, check all loops at every boundary level, and use the software's internal data structures.

However, the inclusion of programming expertise in the testing procedure makes white box testing a complex procedure [9] [10].

Black Box testing is a type of testing that essentially tests the application's functionality without going into the specifics of the implementation. This method can be used at any stage of the SDLC's testing process. It mostly carries out the testing in such a way that it examines each and every feature of the application to see if it satisfies the user's initial requirements. By testing their functionality at each minimum, maximum, and base case value, it can identify incorrect functions. It is the most common and straightforward method of testing worldwide [9, 10].

III. Software Testing Life Cycle

The Test Plan is a mandatory document that is geared toward the application's functional testing; without it, the testing process is impossible [11].

The test case is created during the test designing phase, which concludes the test planning process. The QA team either writes appropriate test cases by hand or, in some instances, creates automated test cases. A set of test inputs or data, the conditions of execution, and the expected outcomes are all specified in a test case. It is important to select the specified set of test data in such a way that it produces both the expected result and data that is intentionally incorrect and will result in an error during the test. Usually, this is done to see under what circumstances the application stops working [11].

The test cases that were developed prior to the execution phase are carried out in the Test Execution Phase. Every failed test case will be linked to the error or bug that was discovered if the functionality passes the execution phase without receiving any bug reports. A defect or bug report is the output of such activity.

After the test cases have been run, the generated results are reported as part of test reporting [11]. Bug reports are also sent to the development team so that they can be fixed.

IV. Software Release Life Cycle

This life cycle follows the STLC and includes additional testing, including Alpha and Beta testing. Alpha Testing, where Alpha stands for the initial developer-level testing of the application, can be carried out using either the white box technique or the grey box technique. A black box approach, also known as an alpha release, could be used for testing at the system or integration level. When a feature freeze occurs, alpha testing ends and no more features are added to improve functionality or serve any other purpose [13] [14].

Because it is performed by the user following the Alpha release, the Beta testing phase can be regarded as formal acceptance testing. It comes after the Alpha testing phase. For the purpose of testing, the software or application is made available to a specific set of intended users. Before an application is officially released, the beta version is typically made available to the intended audience for feedback. The intended audience is frequently referred to as Beta Testers, and the application may be referred to as a prototype software version primarily for demonstration purposes.

As a result, the software's final version is released following beta testing [15, 16].

V. Enhancement Of The Testing Processes

Test Suite Prioritization uses Combinational Criteria to improve the testing procedure. The primary approach to this kind of test case prioritization is to convert the weblogs into test suites that are relevant to the user session and then write them down in an XML format. Coverage based on combinatorial test suites should accurately prioritize the algorithm used in this method. In addition, empirical studies ought to be carried out in order to evaluate the efficiency of the particular application and the relevant test suites it contains [17]. C-PUT, a tool used in this manner, basically formats the logs of web applications into XML-formatted test suites; The functionality for

prioritizing these tests is then made available through its use.

The question of whether or not these test suite prioritization methods can be utilized to improve the fault detection ratio is the subject of ongoing research [18, 19]. Another improvement to the software testing process is the use of genetic algorithms (GAs) to automate the generation of test data for the application. Previously, the use of dynamic methods for the generation of test data remained a significant issue; genetic algorithm-based testing, on the other hand, is not only effective for the generation of test data but also capable of handling the generation of data in accordance with the degree of program complexity [20].

VI. Test Automation

Test automation which is the utilization of particular software to carry out the testing process and also makes the comparison of actual results with the expected results, is the most significant improvement to the testing process.

The Test Automation technique saves time by eliminating the need for laborious manual testing.

Test Automation is used in the SDLC both during the testing phase and the implementation phase. Because it saves a significant amount of time and completes the testing processes in a shorter amount of time, Test Automation is being used worldwide in place of manual testing. By reducing the need for manual testing and revealing the number of errors and deficiencies that cannot be identified through manual testing, test automation has replaced manual testing.

One of the most common types of testing, regression testing, takes a lot of time when done manually. After any bugs or errors have been fixed, it typically checks to see if the software or application functions properly. because the error or bug ratio of the code or application occasionally rises even higher after the error is fixed. Therefore, to avoid the time required for regression testing, For this purpose, a collection of automated test suites is assembled into a regression test suite. Additionally, Test Automation aids in the discovery of the issue at a much earlier stage, saving a great deal of time and money on modifications in subsequent phases [21].

VII. Testing Frame Works

Testing framework in the Agile Lifecycle Another innovation in software testing is the agile lifecycle, which includes short, quick test cycles and frequently changing requirements. Therefore, any testing framework can be used in an agile environment; however, maintaining a test automation suite becomes quite challenging because of the frequent iterations and rapid changes in specified requirements.

However, testing frameworks do not work well in an agile environment because it is difficult to cover as much code and functionality.

Test Driven Development (TDD) is a method that uses automated unit tests to force the decoupling of dependencies and drive the design of software. TDD provides a crystal clear measure of success when the test no longer fails, boosting confidence that the system satisfies its core specifications. In contrast to the conventional testing process, which frequently results in the discovery of one or more errors or defects, TDD does not. When the TDD method is used, a significant amount of time that might otherwise be wasted during the debugging process can be saved [21].

Behaviour Driven Development, or BDD, is primarily an extension of Test-Driven Development that focuses on the system's behavior rather than the implementation level. As a result, increasing the efficiency of the testing process is made possible by providing a clear understanding of the system's intended function. As a result, BDD is primarily Test-driven Development combined with acceptance testing, which typically entails testing to determine whether a product or software requirement is met or not. User Acceptance Testing is the term for it if it is carried out by the intended customer or user [22].

VIII. Metrics For Testing

A. Metrics for Prioritization The use of Test Metrics is crucial because they have the potential to significantly boost the efficiency of the testing process.

They are a crucial indicator of the accuracy, efficiency, and analysis of specified metrics. They may also be of assistance in determining the areas that need to be improved and the subsequent action or step that must be taken to eliminate them. Software Testing Metrics focus on the quality aspects relevant to the process and product and are divided into Process Quality Metrics and Product Quality Metrics, both of which aim to provide enhancements in both the testing process and the quality of the product [23, 24]. Test Metrics are not just a single step in STLC; rather, they serve as an umbrella for the continuous improvement of the entire testing process itself [23, 24].

However, the matching of the testing approach to the application under development is a critical issue for the existing testing process. In every application that needs to be developed, not every testing strategy can be used. For instance, the testing of a network protocol software will be very different from testing a specific e-commerce application because the complexity of the test cases will be very different. This 180 demonstrates how important it is to involve humans in the testing process rather than just relying on the existing test cases. The number of HTTP requests in a test case determines the length of the test, which is one of the priority metrics. Frequency-based prioritization improves the testing process by prioritizing the test cases that use the most frequently used pages over those that use less frequently used pages [25] [26].

B. Process Quality Metrics A process is the most important part because it can produce a high-quality product in the shortest amount of time and at the lowest cost. This is the main reason why businesses all over the world have focused on making processes work better. This is also where the need for metrics came from because metrics are needed to accurately measure a process from a variety of angles.

The most important metric for process quality is process efficiency, which includes things like the Test progress Curve, which shows how far along the test plan the Testing Phase is expected to go [27] [28].

The next important step in the metric, both in terms of the phases and the components, is the cost of testing. The primary objective is to assist in determining which parts will require extensive testing and how much they will pay for it. Another metric that shows how long it takes the testing team on average to verify defects is the average defect turnaround time. An indicator of operational efficiency is the metric known as Average Defect Response Time. It is the average amount of time it takes the team to correct errors.

Metrics for Process Effectiveness guarantee that the application or products that are produced will be of high quality. Major components of it include test coverage, defect removal efficiency, the requirement volatility index, failed and completed test cases, and ensuring an improved testing process overall. Additionally, since RTM (Requirement Traceability Matrix) maps each test case with a specific requirement, it can improve it.

IX. Objectives Of Software Testing Applied

Detection All of the system's deficiencies, errors, flaws, and failures are discovered through detection.

Detection defines the system's capabilities and limitations, as well as the quality of each system component, the products that work, and the system as a whole. Testing must be carried out several times on the system with incorrect input to ensure that what will happen and what should not will be detected.

B. Prevention This information provides details on how to prevent errors and decrease the number of flaws in order to clarify the system's overall specification and required performance. It identifies the factors that lead to errors and helps us avoid them in the future.

C**. Improvement of quality**:

"**Quality cannot** be achieved by evaluating a product that has already been completed." [1] A software system's poor quality and unreliability can kill and cause disasters in critical environments. If there are bugs in the system

An efficient test helps to reduce the number of errors and ensures an increase in software quality.

D. Confirmation:

to make sure the software has the necessary features. At the beginning of the development process, verification is completed. The fulfillment of specified requirements is the primary objective of verification. In essence, verification asked, "Are we building the product correctly?"

E. Checking:

Validation guarantees that we are evaluating the appropriate software in accordance with the user's instructions. Validation can be carried out either at the beginning or end of the development process. The question posed by validation was, "Are we building the right product?"

X. Related Work

Testing software is an important part of the software development life cycle. It assures us that the system will function as required. Various software testing methods and strategies are utilized for this purpose. White box, black box, and grey box testing are these methods. The tester tests the requirements and the final result with a black box. The tester tests the internal design and source code using a white box.

Gray box testing combines the advantages of black box and white box testing.

Unit testing is testing only a small number of units. Integration testing is used to test various integrated modules.

System testing, on the other hand, is used to test the system as a whole.

In this paper, we compare and discuss these strategies and techniques to determine which one is appropriate for which situation. We go into great detail about them.

XI. Methods For Testing Software

A. Tests with black boxes:

a method for testing software that plays a significant role in testing software. Black box testers do not have access to source code or any knowledge of the internal design when they are testing. Testers only have system architecture knowledge. This method is intended to guarantee that all system inputs are accepted in the specified manner and that the output is correct.

[2] The following are the most well-known black box software testing methods.

1) Analysis of boundary values:

The boundary failure of the system is a possibility. because programming at the edges of equivalence classes increases the likelihood of error. This is why this method focuses on choosing values or edges at extreme boundaries.

2) Partitioning by equivalence:

We are able to reduce the number of test cases thanks to these methods. This method basically divides the program's input domain into equivalence classes based on input values. These input domain-derived equivalence classes are used to create test cases.

3) Testing with Orthogonal Arrays:

Orthogonal array testing is a statistical method of testing that helps reduce the number of test combinations and is used when the input domain is very small. Columns are used to represent variables, and rows are used to represent test cases.

4) Diffusing:

Barton Miller invented the black box testing method known as "fuzzing" in 1989 at the University of Wisconsin. The application is fed random input in this method. Implementation flaws can be identified through fuzzy testing, which employs malformed or partially malformed data injection in automated or semi-automated session fuzz tests.

5) Testing based on graphs:

Black box testing method that begins with the creation of a graph. The input modules are used to create the graph. An identifier is given to input modules. Through graph a connection is established between effect and its causes. 6) All Pair Testing: A sort of black box technique in which the purpose of test cases is to execute all discrete combinations which are possible for input parameter of each pair.to cover all the pairs we need to use a number of test cases. 7) State Transition Diagrams (or) State Graphs: A brilliant tool which is used to capture several types of system requirements and documented internal system design. This tool is also used to test state machine and to navigate GUI (graphical user interface).

Tests in white boxes:

Structural Testing and Code-Based Testing are two other names for this method [6, 7, 8, 9, 10, 11, 12]. This testing method focuses primarily on examining the internal logic and structure of source code. It can be used at the integration, unit, and integration levels.

A tester must have complete knowledge of the source code in order to use this technique [13, 14, 15, 16].

The following are the most well-known white box software testing methods.

1) Checking the desk:

Desk checking is a manual method for verifying programs' logic. Typically, the programmer conducts this test and records the results with a pen and paper.

2) A Guide to the Code:

It is a type of peer review in which the programmer leads the review process and asks other team members questions about the system to find errors.

3) Inspection in writing:

This formal, cost-effective, and effective method is used to find design and code errors. supervised by an outside contractor. The most formal method is this one. Its primary objective is to find errors, side effects, and violations.

4) Testing of Control Flow:

a fundamental approach that works for all software. This is used with almost all software and is based on code coverage structure, which refers to the number of

programs that have been tested. 100% coverage is the technique's criterion and goal.

5) Testing the Basis Path:

This method will be utilized to assess procedural design's logical complexity [17, 18, 19]. Each distinct path of code is tested separately for this purpose. In the program, control flow will be represented by flow graphs.

6) Testing the data flow [20, 21, 22]:

The control flow graph can be used to learn how this method defines and uses program variables.

7) Testing a Loop:

Errors in loops typically occur near the beginning and end. The validity of the loop construct—is there a possibility that the loop will successfully terminate—is the primary focus of this method.

Testing in a gray box:

This is a testing method in which little is known about the internal application work. This method works on any platform or language.

Grey box testing combines the advantages of black box and white box testing. It uses algorithms and internal data structure to design test cases less frequently than white box testing, but more frequently than black box testing.

Grey box is the combination of a white box and a black box. This method will be used to test a piece of software against its specification without knowing everything about how it works inside. It is frequently used in integration testing, but it can also be used in most phases of testing.

The following are the most well-known grey box software testing methods.

1) An oblique array:

Utilizing a subset of all possible combinations, this testing strategy This is statistical and systematic.

2) Testing matrices:

The project static report is stated using this method.

3) Testing for Regression:

Running test cases is necessary for regression testing. if the system is to undergo new changes.

4) Testing Patterns:

to make sure the application's architecture and design are good. This method will be utilized.

The following is a list of various integration testing techniques.

- **Test of Module Interface:**

This method is used to determine whether or not the flow of information into the program unit and out is correct.

- **2) Local data models:**

This is used to determine whether the temporary data are properly stored and maintain their integrity.

- **3) Boundary circumstances:**

to make sure that the module boundary conditions are met so that the program can run properly at boundaries.

- 4**) Alternative routes:**

to guarantee that each independent path is tested at least once after each module's statements have been executed.

- **5) Paths for handling errors:**

After each test has been successfully completed, error handling paths must be used to ensure that errors are handled appropriately.

Integration Testing Integration testing is the process of merging software modules and testing them as a group. Between unit testing and system testing is this testing.

This method simultaneously builds program structure and uncovers errors. The interface between modules or units is the purpose of this method.

The following is a list of various integration testing techniques.

- **Integration from the top:**

This incremental approach is used to build program structure. The procedure repeats itself until all modules have been integrated and tested, starting with the most advanced module.

- **Integration from the bottom up:**

Test the application as a whole as single unit integration continues until all modules are integrated, starting with the application's innermost unit.

- **3) Big bang experiments:**

This is not an incremental strategy. Start by combining all of the parts, then test the program as a whole and a set of errors. Because errors occur after

they are corrected, this method can be used indefinitely.

Testing A System:

When a system meets its specified requirements, this kind of testing is carried out. This testing does not necessitate familiarity with source code. This is a collection of various tests. This kind of testing aims to fully test the application. The purpose of each test on the system is different. This verifies that the system is capable of carrying out required functionality and is properly integrated.

The following are different methods for testing a system:

1) Recovery testing:

Non-functional testing is one type. The primary objective is to compel the software to fail to verify how quickly and effectively recovery is carried out. Re-initialization, restart, and data recovery are evaluated if recovery is automated to ensure that the system check pointing mechanisms are functioning correctly.

2) Testing for security:

It ensures that there are no loopholes that result in significant losses. This test aims to uncover all system flaws and loopholes. Testers use customer software designed to break the system's defenses to attack the system.

3) Testing of graphical user interfaces:

This method is used to test the user's graphic user interface to make sure it meets the requirements. Testing screen controls like menus, toolbars, buttons, windows, and dialog boxes, among others, is part of this.

4) Testing for compatibility:

This testing is not functional. By running the software on various operating systems, networks, hardware, browsers, and databases, compatibility is checked.

XII. Conclusion Work

This paper explains what software testing is and what its methods and strategies are. Software testing is the process of running a software system to look for errors and make sure it meets the requirements.

Provide an independent perspective on the system. Risk, which may arise from the implementation of software, enables the company to appreciate and comprehend risk. When testing is based on user requirements and for large code segments, the black testing technique should be used. In order to get rid of extra lines of code that aren't needed, white box testing is used to find implementation and internal code errors. The testing of interface and functional specification is done with grey box testing. Software and hardware units that are either separate units or related units that are grouped together are subject to unit testing.

The purpose of integration testing is to evaluate the interaction between software and hardware components, as well as combinations thereof. The purpose of system testing is to evaluate the system as a whole. We compare and elaborate on various strategies and methods.

such as the Black-Box, White-Box, and Grey-Box tests. System testing, integration testing, and unit testing are also compared.

Because the product's final delivery is dependent on testing, it is the most important step in the Software Development Lifecycle. Because it is a laborious and time-consuming process, improved methods and novel approaches are required. This makes it possible to use automated testing and other test metrics both before and during the testing process. It can improve the testing methods that are already in place, making them more efficient in terms of time and producing a reliable and effective product that not only meets the specified requirements but also maximizes operational efficiency.

The platform on which software development and testing are carried out continues to grow in importance. Testing, on the other hand, is an extremely important and significant step that typically occurs quite late in the software development process.

For better understanding and early review, specification writers and testers should interact as much as possible. This could eliminate ambiguity issues and save money on software maintenance in the future. Before releasing the project for official testing, testers should provide developers with a specific lightweight test model so that they can verify that the primary specifications have been met.

Exception testing and methods for handling exceptions can be best determined with the help of simulation tools, which can greatly assist testers in creating a similar environment to that of the product. By incorporating simulation into the testing process, the product can be tested in a similar testing environment to that intended for the product. As a

result, the work that is relevant to the testing process in the future will be much more dependent on technology, using a model-based simulation and automated testing approach to not only speed up the testing life cycle but also provide optimal bug prevention and effective quality assurance.

XIII. References

[1] P. Ron. Software testing. Vol. 2. Indianapolis: Sam's, 2001.

[2] S. Amland, "Risk-based testing:" Journal of Systems and Software, vol. 53, no. 3, pp. 287–295, Sep. 2000.

[3] Redmill and Felix, "Theory and Practice of Risk-based Testing", Software Testing, Verification and Reliability, Vol. 15, No. 1, March 2005.

[4] B. Agarwal et al., "Software engineering and testing". Jones & Bartlett Learning, 2010.

[5] K. Bogdan. "Automated software test data generation". Software Engineering, IEEE Transactions on 16.8 (1990): 870-879.

[6] Jacobson et al. The unified software development process. Vol. 1. Reading: Addison-Wesley, 1999.

[7] Everett et al., "Software testing: testing across the entire software development life cycle". John Wiley & Sons, 2007.

[8] J.Irena. "Software Testing Methods and Techniques", 2008, pp. 30-35

[9] Guide to the Software Engineering Body of Knowledge, Swebok, A project of the IEEE Computer Society Professional Practices Committee, 2004.

[10] E. F. Miller, "Introduction to Software Testing Technology", Software Testing & Validation Techniques, IEEE, 1981, pp. 4-16.

[11] M. Shaw, "Prospects for an engineering discipline of software," IEEE Software, November 1990, pp.15-24.

[12] D. Nicola et al. "A grey-box approach to the functional testing of complex automatic train protection systems." Dependable Computing-EDCC 5. Springer Berlin Heidelberg, 2005. 305-317.

[13] J. A. Whittaker, "What is Software Testing? And Why Is It So Hard?" IEEE Software, 2000, pp. 70-79.

[14] N. Jenkins, "A Software Testing Primer", 2008, pp.3-15.

[15] Luo, Lu, and Carnegie, "Software Testing Techniques Technology Maturation and Research Strategies', Institute for Software Research International-Carnegie Mellon University, Pittsburgh, Technical Report, 2010.

[16] M. S. Sharmila and E. Ramadevi. "Analysis of performance testing on web application." International Journal of Advanced Research in Computer and Communication Engineering, 2014.

[17] S. Sam path and R. Bryce, Improving the effectiveness of Test Suite Reduction for User-Session-Based Testing of Web Applications, Elsevier Information and Software Technology Journal, 2012.

[18] B. Pedersen and S. Manchester, Test Suite Prioritization by Cost based Combinatorial Interaction Coverage International Journal of Systems Assurance Engineering and Management, SPRINGER, 2011.

[19] S. Sprenkle et al., "Applying Concept Analysis to User-session based Testing of Web Applications", IEEE Transactions on Software Engineering, Vol. 33, No. 10, 2007, pp. 643 – 658

[20] C. Michael, "Generating software test data by evolution, Software Engineering", IEEE Transaction, Volume: 27, 2001.

[21] A. Memon, "A Uniform Representation of Hybrid Criteria for Regression Testing", Transactions on Software Engineering (TSE), 2013.

[22] R. W. Miller, "Acceptance testing", 2001, Data retrieved from(http://www.dsc.ufcg.edu.br/~jacques/cursos/map/recursos/Testin g05.pdf)

[23] Infosys, "Metric model", white paper, 2012. Data retrieved from , ijsaet,vol25, issue5,pp-78-89.

[24] B. Boehm, “Some Future Trends and Implications for Systems and Software Engineering Processes”, 2005, pp.1-11.

[25] R. Bryce, “Test Suite Prioritization and Reduction by Combinational based Criteria”, IEEE Computer Society”, 2014, pp.21-22.

[26] M. I. Babar, “Software Quality Enhancement for value based systems through Stakeholders Quantification”, 2005, pp.359-360. Data retrieved from (http://www.jatit.org/volumes/Vol55No3/10Vol55No3.pdf)

[27] R. Ramler, S. Biffl, and P. Grünbacher, "Value-based management of software testing," in Value-Based Software Engineering. Springer Science Business Media, 2006, pp. 225– 244.

[28] D. Graham, "Requirements and testing: Seven missing-link myths," Software, IEEE, vol. 19, 2002, pp. 15-17