International Journal For Technological Research In Engineering
Volume 3, Issue 5, January-2016 *ISSN (Online): 2347 - 4718*

A BRIEF OVERVIEW OF SOFTWARE TESTING TECHNIQUES AND TEST AUTOMATION

Divya C D[1], Bharath T S[2]
[1]Assistant Professor, Dept of Computer Science and Engineering, GSSS Institute of Engineering and Technology for Women, Mysuru, India, Affiliated to VTU Belgaum
[2]PG Student at SS College of Engineering, Udaipur, India

Abstract: Software testing is the process used to measure the quality of developed computer software. It exhibits all mistakes, errors and flaws in the developed software. Software testing is the process of evaluating the developed system to assess the quality of the final product. Unfortunately, software-testing process is expensive and consumes a lot of time through software development life cycle. As software systems grow, manual software testing becomes more and more difficult. Therefore, there was always a need to decrease the testing time. Recently, automation is as a major factor in reducing the testing effort by many researchers. Therefore, automating software-testing process is vital to its success.
Keywords: Black box, Demonstration, Detection, Grey box, Prevention, Software testing, White box, Software Testing; Automated Software Testing; Test Data; Test Case; Test Script; Manual Testing; Software Under Test; Graphical User Interface.

I. INTRODUCTION

Software testing has evolved since 1970's as an integral part of software development process. Through it, the final quality of the software can be improved by discovering errors and faults through interacting, checking behavior and evaluating the System Under Test (SUT) to check whether it operates as expected or not on a limited number of test cases with the aim of discovering errors that are found in the software and fixing them. According to Ilene Burnstein, software testing describes as a group of procedures carried out to evaluate some aspect of a piece of software [1]. Ehmer Khan [2] shortly defines it as a set of activities conducted with the intent of finding errors in software. In addition, according to Ammann and Offutt [3] software testing means evaluating software by observing its execution. Since software-testing process is a very expensive process, complete testing is practically impossible and it is not acceptable to reduce testing effort by accepting quality reductions. Testing effort is often a major cost factor during software development. Many software organizations are spending up to 40% of their resources on testing [4]. Therefore, an existing open problem is how to reduce testing effort without affecting the quality level of the final software. Automation is one major solution for reducing high testing effort. Automating certain manual tasks from software testing process can save a lot of testing time. It can help in performing repetitive tasks more quickly than manual testing.

II. OBJECTIVE OF TESTING

The objective of testing is to find problems and fix them to improve quality (Fig.1). Software testing typically represents 40% of a software development budget.
There are four main objectives of software testing:

- Demonstration: It demonstrates functions under special conditions and shows that products are ready for integration or use.
- Detection: It discovers defects, errors and deficiencies. It determines system capabilities and limitations, quality of components, work products and the system.
- Prevention: It provides information to prevent or reduce the number of errors clarify system specifications and performance. Identify ways to avoid risk and problems in the future.
- Improving Quality: By doing effective testing, we can minimize errors and hence improve the quality of software. [5]

III. DIFFERENT TESTING TECHNIQUES

A. Black Box Testing

Black Box Testing is based on the requirements specifications and there is no need to examining the code in black box testing. This is purely done based on customers view point only tester knows the set of inputs and predictable outputs.

- Equivalence Partitioning: This technique divides the input domain of a program into equivalence classes from which test cases can be derived, so it can reduce the number of test cases.
- Boundary Value Analysis: It focuses on testing at boundaries, or where the extreme boundary values are chosen. It includes minimum, maximum, just inside/outside boundaries, error values and typical values.
- Fuzzing: This technique feeds random input to application. It is used for finding implementation bugs, using malformed/semi-malformed data injection in an automated or semi-automated session.
- Cause-Effect Graph: In this technique, testing begins by creating a graph and establishing the relation between effect and its causes.
- Orthogonal Array Testing: It can be applied where input domain is very small, but too large to

accommodate exhaustive testing.

All Pair Testing: In this technique, test cases are designed to execute all possible discrete combinations of each pair of input parameters. Its main objective is to have a set of test cases that covers all the pairs.

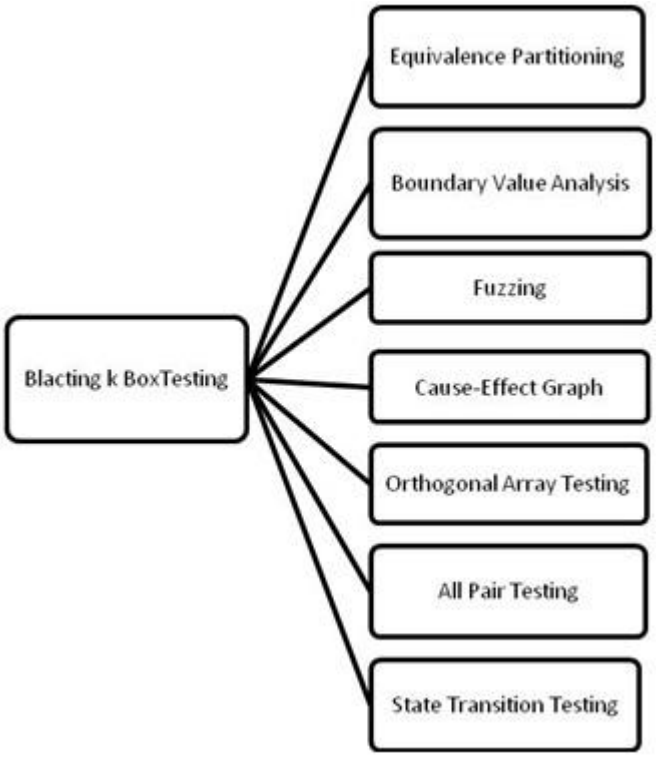

Fig.1 Represent different forms of black box testing

- State Transition Testing: This type of testing is useful for testing state machine and also for navigation of graphical user interface.

Advantages:

- Testers need not to have knowledge on specific programming language.
- Testing is done from user's point of view.
- It helps to expose any ambiguities or inconsistencies in the requirement specifications.[6]
- Programmer and tester both are independent of each other.

Disadvantages:

- Test cases are hard to design without clear specifications.
- Chances of having repetition of tests that are already done by programmer.
- Some parts of back end are not tested at all.

B. White Box Testing

White box testing mainly focuses on internal logic and structure of the code. White-box is done when the programmer has techniques full knowledge on the program structure. With this technique it is possible to test every branch and decision in the program.[7][8]

- Desk Checking: Desk checking is the primary testing done on the code. The authors who have knowledge in the programming language very well will be involved in desk checking testing.
- Code Walkthrough: In this testing process a group of technical people go through the code. This is one type of semi-formal review technique.
- Formal Inspections: Inspection is a formal, efficient and economical method of finding errors in design and code. It's a formal review and aimed at detecting all faults, violations and other side effects.

Control Flow Testing: It is a structural testing strategy that uses the program control flow as a model control flow and favors more but simpler paths over fewer but complicated path.

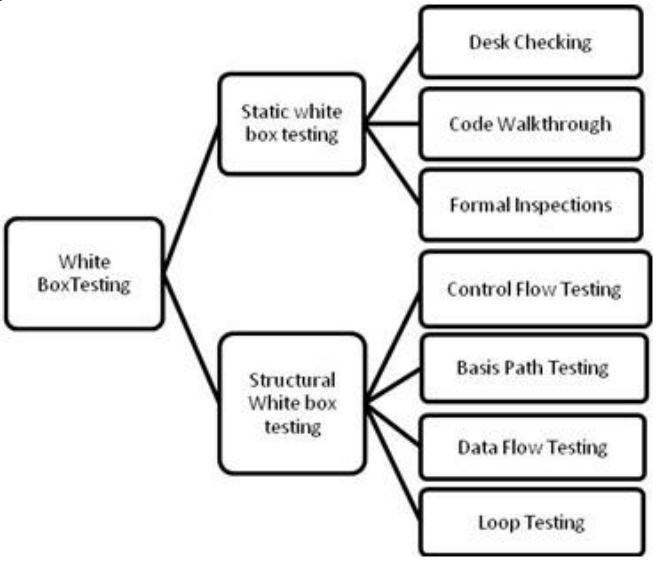

Fig. 2 Represent different forms of white box testing

- Basis Path Testing: Basis path testing allows the test case designer to produce a logical complexity measure of procedural design and then uses this measure as an approach for outlining a basic set of execution paths.
- Data Flow testing: In this type of testing the control flow graph is annotated with the information about how the program variables are define and used.
- Loop Testing: It exclusively focuses on the validity of loop construct.

Advantages:

- It reveals error in hidden code by removing extra lines of code.
- Maximum coverage is attained during test scenario writing.[9]
- Developer carefully gives reasons about implementation.

Disadvantages:

- A skilled tester is needed to carry out this testing because knowledge of internal structure is required.
- Many paths will remain untested as it is very

difficult to look into every nook and corner to find out hidden errors.

C. Grey Box Testing:

Gray-box testing attempts, and generally succeeds, to combine the benefits of both black-box and white-box testing. Gray-box testing takes the straight-forward approach of black-box testing, but also employs some limited knowledge of the inner workings of the application.

White box + Black box = Grey box,

it is a technique to test the application with limited knowledge of the internal working of an application and also has the knowledge of fundamental aspects of the system. [7] Therefore, a tester can verify both the output of the user interface and also the process that leads to that user interface output. Gray-box testing can be applied to most testing phases; however it is mostly used in integration testing.

- Orthogonal Array Testing: This type of testing use as subset of all possible combinations.
- Matrix Testing: In matrix testing the status report of the project is stated.
- Regression Testing: If new changes are made in software, regression testing implies running of test cases.

Pattern Testing: Pattern testing verifies the good application for its architecture and design.

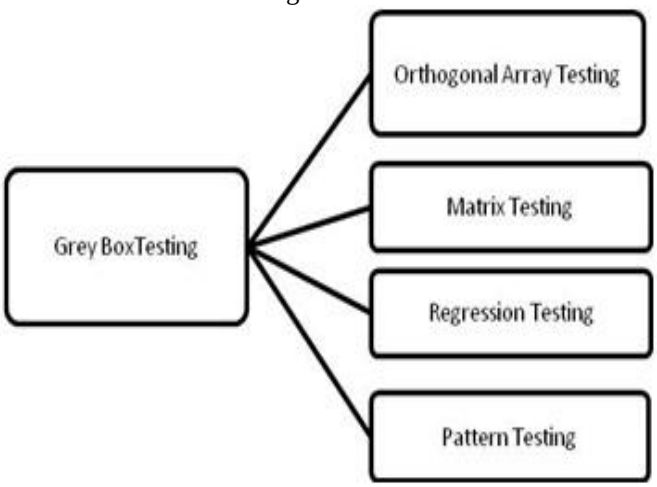

Fig. 3 Represent different form of grey box testing

Advantages:

- It provides combined benefit of black box and white box testing techniques.
- In grey box testing, tester can design excellent test scenarios.
- Unbiased testing
- Create an intelligent test authoring.

Disadvantages:

- Test coverage is limited as the access to source code is not available.
- Many program paths remain untested.
- The test cases can be redundant.[7]

TABLE 1
COMPARISON BETWEEN THREE FORMS OF TESTING TECHNIQUES

Sr. No.	Black Box Testing	White Box Testing	Grey Box Testing
1.	Analyses fundamental aspects only i.e.no knowledge of internal working.	Full knowledge of internal working.	Partial knowledge of internal working.
2.	It is least exhaustive and time consuming.	Potentially most exhaustive and time	It is somewhere in between the two.
3.	Not suited for algorithm testing.	It is suited for algorithm testing(suited for all).	Not suited for algorithm testing.
4.	Granularity is low.	Granularity is high.	Granularity is medium.
5.	Performed by end users and also by tester and developers(user acceptance testing).	It is performed by developers and testers.	Performed by end users and also by tester and developers (user acceptance testing).

IV. TEST AUTOMATION

Test automation is the use of software (under a setting of test preconditions) to execute tests and then determine whether the actual outcomes and the predicted outcomes are the same. For example, Windows Vista offers per-application volume. It is possible to turn down the volume on a game while leaving Windows Media Player playing loud. To do this, right-click on the speaker icon in the lower right-hand corner of your screen and select "Open Volume Mixer." Moving the slider for an application down should cause its volume to decrease. Testing this manually is easy. Just play a sound, lower the volume, and listen. Now automating this rather than doing it manually is the process of test automation. Companies not only want to test software adequately, but also as quickly and thoroughly as possible. To accomplish this goal, organizations are turning to automated testing. To increase the test coverage Reduces the need for manual testing and discovers defects manual testing cannot expose and also manual testing is error prone and a time consuming process. Running the tests again and again gives us the confidence that the new work we added to the system did not break the code that used to work and also to make sure that the changes we introduced are working. Executing the tests (particularly acceptance tests) can also help us understand what portion of the desired functionality has been implemented. The set of the automated test suite can form a regression test suite. The purpose of the regression suite is to make sure that the software behavior is unchanged unless due to data change or latest software. Automating also reduces the time taken for regression testing. Automated unit test suite helps find the problems at an earlier stage and solve them. If we have a group of testers and suppose if each project implements a unique strategy then the time needed for the tester become productive in the new environment will take long. To handle this we cannot

make changes to the automation environment for each new application that comes along. For this purpose we use a testing framework that is application independent and has the capability to expand with the requirements of each application. Also an organized test framewps in avoiding duplication of test cases automated across the application. In short Test frameworks helps teams organize their test suites and in turn help improve the efficiency of testing.

Modular Testing Framework: The Modularity testing framework is built on the concept of abstraction. This involves the creation of independent scripts that represent the modules of the application under test. These modules in turn are used in a hierarchical fashion to build large test cases. Thus it builds an abstraction layer for a component to hide that component from the rest of the application. Thus the changes made to the other part of the application do not effect that component.
Advantages: Modular division of scripts leads to easier maintenance and also the scalability of the automated test suites. The functionality is available in easy to use test libraries so creating new driver scripts for different tests is easy and fast.
Disadvantages: The main problem with modular frameworks is that the test script have test data embedded in them. So when the test data needs to be updated we need to change the code of the script. This becomes a big problem when the test script is large. For this purpose, data- driven testing frameworks have been introduced.

Data-Driven Testing Framework: Data driven testing is where the test input and the expected output results are stored in a separate data file (normally in a tabular format) so that a single driver script can execute all the test cases with multiple sets of data. The driver script contains navigation through the program, reading of the data files and logging of the test status information.

Advantages: This framework reduces the number of overall test scripts needed to implement all the test cases. Less amount of code is required to generate all the test cases. •Offers greater flexibility when it comes to maintenance and fixing of bugs. The test data can be created before test implementation is ready or even before the system to be tested is ready.
Disadvantages: The test cases created are similar and creating new kind of tests requires creating new driver scripts that understand different data. Thus the test data and driver scripts are strongly related that changing either requires changing the other. For this purpose keyword driven testing frameworks have been introduced.

Keyword- Driven Testing Framework: Keyword driven testing is an application independent framework utilizing data tables and self explanatory keywords to explain the actions to be performed on the application under test. Not only is the test data kept in the file but even the directives telling what to do which is in the test scripts is put in external input data file. These directives are called keywords. The keyword based testing is an extension to the data driven testing.

Advantages: It has all the advantages that data driven testing has. Automation expertise is not required to maintain or create a new set of test cases. Keywords are reused across multiple test cases.
Disadvantages: The main problem is that this requires a more complicated framework than the data driven framework. With the keyword driven approach the test cases get longer and complex and this is due to the greater flexibility that this approach offers. So in order to combine the strengths of all the frameworks and mitigate their weaknesses we use the hybrid testing framework.

Hybrid Testing Framework: Hybrid testing framework is the combination of modular, data-driven and keyword driven testing frameworks. This combination of frameworks helps the data driven scripts take advantage of the libraries which usually accompany the keyword driven testing.

V. TEST DRIVEN DEVELOPMENT & BEHAVIOR DRIVEN DEVELOPMENT

Test driven development is a technique of using automated unit tests to drive the design of software and force decoupling of dependencies. With traditional testing a successful test finds one or more defects. But using TDD we have a clear measure of success when the test no longer fails. Thus TDD increases our confidence that the system meets the requirements and that the system is working properly when compared to the confidence that traditional testing provides. Behavior driven development is an extension to the test driven development in that it focuses on the behavior stem rather than the implementation aspect of the system. Thus it gives a clear understanding of what the system should do to both the developer as well as the customer making the testing process even more efficient.

VI. CONCLUSION

Software testing is the activity that executes software with an intention of finding errors in it. Software testing can provide an independent view of the software to allow the business to appreciate and understand the risk of software implementation. To carry out software testing in a more effective manner, this paper provides a comparative study of three main techniques of software testing and Frameworks.

REFERENCES

[1] Burnetein, "Practical Software Testing: process oriented approach," Springer Professional Computing, 2003.
[2] M. E. Khan, "Different Forms of Software Testing Techniques for Finding Errors," International Journal of Software Engineering (IJSE), vol. 7, no. 3, 2010.
[3] P. Ammann and J. Offutt, Introduction to Software Testing, New York: Cambridge University Press,

2008.
[4] F. Elberzhager, A. Rosbach, J. Münch and R. Eschbach, "Reducing test effort: A systematic mapping study on existing approaches," Information and Software Technology 54, p. 1092–1106, 2012.
[5] Software Testing. Gregory M. Kapfhammer. The Computer Science and Engineering Handbook, CRC Press. May, 2004.
[6] Mohd. Ehmer Khan, "Different Approaches to Black Box Testing Technique for Finding Errors," IJSEA, Vol. 2, No. 4, pp 31-40, October 2011
[7] Jovanovi?, I. "Software Testing Methods and Techniques. " The IPSI BgD Transactions on Internet Research: 30
[8] F. Saglietti, N. Oster, and F. Pinte, "White and grey-box verification and validation approaches for safety-and security-critical software systems," Information Security Technical Report, vol. 13, no. 1, pp. 10–16, 2008.
[9] 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.

