ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 4, Issue 11, November 2017

Review on Software Testing Techniques

[1]Chaitali Bhowmik,

[1]Department of Computer Science and Engineering, Galgotias University, Yamuna Expressway Greater Noida, Uttar Pradesh

[1]chaitali.bhowmik@Galgotiasuniversity.edu.in

Abstract: Software testing is the method of running an application in order to find bugs (errors or other defects) in the software. Demand for software applications has pushed developed software quality assurance toward new heights. It has been seen as the most critical stage of the life cycle of software development. Testing will test the software item to assess the difference between real and specified conditions and to evaluate the software features. Software testing results in a minimization of defects and a reduction in software costs. Software testing is an unavoidable part of the life cycle of Software Development, and keeping its criticality in the process of pre- and post-development makes it something that should be catered for with enhanced and efficient methods and techniques. In order to reduce errors, maintenance and overall software costs, software testing is important. One of the main problems in the area of software testing is how to get a proper set of test cases to test a software system. The aim of this paper is to study various and improved software testing techniques for better quality control purposes.

Keywords: Automation Testing, Software Testing Life Cycle, Testing Frameworks, Testing Methodologies.

INTRODUCTION

Testing is known as an evaluation process that either the particular system meets or fails to meet its originally stated requirements. It is mainly a process that includes the validation and testing phase whether the system developed meets the user-defined requirements[1]. This activity thus results in a difference between the actual results and the expected results. Software Testing refers to bug tracking, errors or missing requirements in software developed. So, this is an examination that gives the stakeholders the exact knowledge about the product's quality. Testing may also be taken as a risk-based practice[2]. The important thing that testers need to learn during the testing process is how to simplify a large number of tests into manageable tests, and how to make wise decisions about the risks that are important to test or not[3].

Fig. 1 illustrates the relation between cost of testing and errors. Fig. 1 clearly shows that the cost of testing all forms, i.e. functional and non-functional, goes up drastically. Then decision making for what to test or eliminate tests will cause a lot of bugs to be overlooked. The successful research aim is to do the optimum amount of testing in order to minimize the extra testing effort. Software testing is a necessary part of quality assurance software according to Fig.1. The value of testing can be viewed from life-critical software testing (e.g., flight control) which can be highly expensive due to the risk of delays in scheduling, cost overruns or cancellation. Testing has some levels and measures according to which the person performing the test varies from level to level[4]. Unit testing, Implementation testing and Application testing are the three basic steps in the software testing. Each of these steps is either tested by the developer of the software or by the quality assurance engineer who is also known as software tester.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 4, Issue 11, November 2017

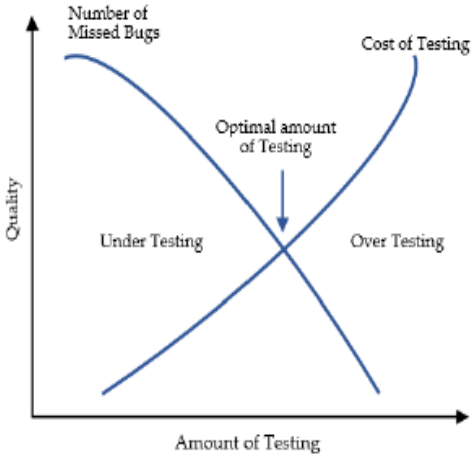

Figure 1: Every Software Project has Optimal Test Effort

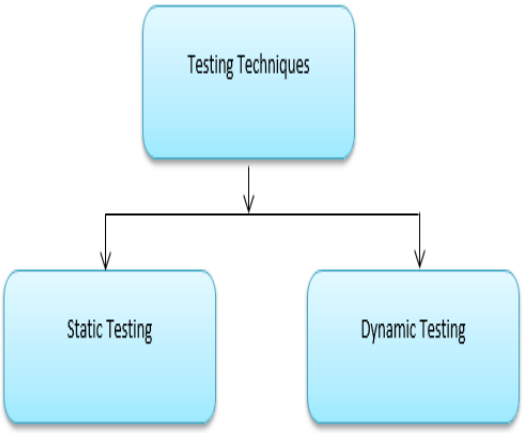

Figure 2: Testing Techniques

SOFTWARE TESTING METHODOLOGIES:

The value of software testing to software quality cannot be overstated. Testing the program to find all the errors is mandatory after code development and they must be debugged before software release. Even if all the errors in the significant software cannot be detected and debugged at every point, it is attempted to remove all the errors as much as possible. Testing helps find the bugs; it is not able to conclude that the software is bug-free[5].
Test techniques generally have two parts (Fig. 2):
Static Testing
Dynamic Testing

Manual Testing (Static Testing):

This applies to the evaluation method where the code is not executed. It does not involve highly qualified professionals as in this phase; the actual implementation of the program is not completed. It starts with the Software Development Life Cycle (SDLC) initial phase; hence it is also known as the verification testing. The main goal of static testing is to improve software product performance by helping software professionals identify and solve their errors early in the process of software development. Documentation such as the Software Requirement Specification (SRS), design documentation, source code, test suites, and web page material are subject to static testing. It is achieved before the code is released. As a result, it provides both code assessment as well as documentation [6].
Techniques for the static testing include:
i. Inspection: Locating defects is done primarily. Moderators conduct the code walkthrough. A checklist is prepared in this type of formal review for checking the working document.
ii. Walkthrough: The process is not formal. That process is being led by the authors. According to his or her thought process, the Author advises the

participants through the document to achieve a common perception. It is particularly useful for documents of a higher level, such as requirement specification, etc.
iii. Technical Reviews: A professional review round is conducted to check whether the code is conducted in accordance with the technical specifications and standards that may include test plans, test strategy, and test scripts.
iv. Informal Reviews: It is the methodology of static testing in which the text is checked unofficially, and useful observations are implemented.
Automated Testing (Dynamic Testing):
Dynamic Testing is a software testing method which analyzes the code's dynamic behavior. In dynamic testing, also known as validation where it considers the actual system. It requires the highly skilled professional having the proper knowledge of the domain. Dynamic testing requires checking the input values of the program and analyzing the output values[7].
Progressive testing is composed of two types:
A. White Box Testing
B. Black Box Testing
C. Grey Box Testing

White Box testing:

Internal system specifications and structures are conspicuously created. So, detecting and solving problems is acutely cost-effective. Bugs are to be detected before triggering bickering. Therefore, they are going to summarize this approach as testing software with its internal structure and coding details. White box testing is also known as precise box analysis or evaluation of the white box or glass box testing or transparent box testing, and structural testing. It's a method to finding errors that the tester has complete data within. In large systems and networks this technique is not used much for debugging. Various types of white box testing involve base path testing, loop testing, testing of the control structure, etc. White-box testing tests the internal structure or functioning of a program, while programming abilities and the system's domestic background are used to design test cases. The tester appoints inputs through the code to apply paths and finalize the appropriate outputs. This is similar to the nodes testing in a circuit. White-box Testing can be used in the software testing process at unit integration and system level. This is commonly done at unit level. Although this testing methodology may reveal several mistakes, it may not recognize the missing specifications and unimplemented parts of the specification [8].
White-box Testing includes these methods:
Application Programming Interface testing tests the application using both public and private APIs by generating checks to satisfy such interface coverage requirements.
Fault Injection Methods – Application of faults to deliberately evaluate the efficacy of test methods.
Tools for code coverage can assess the integrity of a test suite created using any method, including black-box testing. This gives the software team the ability to check the parts of a system that are rarely tested, and ensures verification of the most important function points.
Function coverage is the approach which informs about the performed functions.
Statement coverage is the approach that says 100 per cent of the number of lines performed to complete the test. This guarantees at least once execution of all application paths or branches. This helps to ensure quality.

Black Box testing:

A black box test is a test in which its user does not know or control internal information and workings. It meets criteria for specifications and output the basic aim is to identify the system's requirements. Black box testing contains very little or no evidence on the system's internal logical structure. Therefore, it only tests the system's basic features. It ensures that each input is accepted appropriately and that outputs are produced correctly. Black-box testing checks the features, without any internal implementation

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 4, Issue 11, November 2017

information. The testers have only an understanding of what the software should be doing, not how it does it. This is done simply according to the feature of the customers. The tester knows only the set of inputs and the particular outputs [9].

Methods for checking the Black Box include:

i. Equivalence Partitioning: This methodology breaks up a program's input domain into analogous groups from which test cases can be derived. Therefore, the number of test cases can be minimized.
ii. Boundary Value Analysis: This addresses boundary checking, or where the extreme boundary values are chosen. This includes minimum, maximum, error and typical values.
iii. Fuzzing: This approach takes random data. In an automated or semi-automated session, it is used to identify application bugs, using malformed or semi-malformed data injection.
iv. Orthogonal Array Testing: The input domain in this technique is limited but too broad for exhaustive testing to handle.
v. Cause-Effect Graph: This testing technique starts with the generation of a graph and the relation between impact and its causes.
vi. All Pair Testing: The main goal is to have a collection of test cases, including all the pairs. Here, test cases are designed to perform any possible discrete combinations of every couple parameters input.
vii. State Transition Testing: This approach to testing is useful for navigating a graphical user interface.

Grey-Box Testing:

Gray box testing is the software testing technique, with limited knowledge of the application's internal structure and design. It is defined as a package of testing software that has some data about its internal logic and underlying code. It uses structures and algorithms for the internal information. This strategy requires integration selection between two or more code modules, written by completely different developers. This technique involves reverse engineering in order to work out the maximum values. The testing of grey boxes is objective and non-intrusive. Grey-box Testing holds the knowledge of internal data structures and algorithms to design experiments when conducting those tests at the user level. The tester does not have full access to source code for the program. Some subtypes of grey-box testing are as follows[10]:
i. State-Model-Testing: It examines each method of object, transition, and transition paths in each state of object.
ii. Class-Diagram Testing: It examines all of the derived classes of the base class.
iii. Sequence-Diagram Testing: It explores all the methods used in the sequence diagram.
iv. Thread-Based Testing: All classes of single Use-Cases are integrated into this approach, and then testing is carried out. This approach continues until all of the courses of all Use-Cases are considered.
v. Use-Based Testing: In this test, the testing is performed on the classes that either need the services from other courses or need no services.

CONCLUSION

Software testing is a method of executing a program to determine whether or not it meets the stated requirement. The design of the test case is carried out, a plan can be developed and the results can be evaluated against the standards prescribed. Testing is the most crucial part of the life cycle of software development, as it is something on which the product's final delivery depends. It is time consuming and a complex process therefore it includes improved technologies and groundbreaking methodologies. It allows implementation of Automated Testing and other different Test Metrics before and during the testing process. It can improve an existing test method for both time efficiency as well as efficient and reliable final product that not only satisfy the defined specifications but also provide optimum operating performance.

REFERENCES

1. A. A. Sawant, P. H. Bari, and P. . Chawan, “Software Testing Techniques and Strategies,” J. Eng. Res. Appl., 2012.
2. S. Abhijit, B. Pranit, and C. P.M, “Software Testing Techniques and Strategies,” Int. J. Eng. Res. Appl., vol. 4, no. 4, pp. 99–102, 2014.
3. D. Bhargava and A. Veda, “A Different Techniques and Strategies for Software Testing,” Int. J. Eng. Sci. Res. Technol., vol. 2, no. 12, pp. 3754–3756, 2013.
4. M. A. Jamil, M. Arif, N. S. A. Abubakar, and A. Ahmad, “Software testing techniques: A literature review,” in Proceedings - 6th International Conference on Information and Communication Technology for the Muslim World, ICT4M 2016, 2017, doi: 10.1109/ICT4M.2016.40.
5. G. J. Myers, T. M. Thomas, and C. Sandler, The Art of Software Testing 3rd Edition. 2011.
6. R. Misra, C. R. Panigrahi, B. Panda, and B. Pati, “Software design,” in Application Development and Design: Concepts, Methodologies, Tools, and Applications, 2017.
7. S. Anand et al., “An orchestrated survey of methodologies for automated software test case generation,” J. Syst. Softw., 2013, doi: 10.1016/j.jss.2013.02.061.
8. M. Ehmer and F. Khan, “A Comparative Study of White Box, Black Box and Grey Box Testing Techniques,” Int. J. Adv. Comput. Sci. Appl., 2012, doi: 10.14569/ijacsa.2012.030603.
9. C. Henard, M. Papadakis, M. Harman, Y. Jia, and Y. Le Traon, “Comparing white-box and black-box test prioritization,” in Proceedings - International Conference on Software Engineering, 2016, doi: 10.1145/2884781.2884791.
10. W. Yang, M. R. Prasad, and T. Xie, “A grey-box approach for automated GUI-model generation of mobile applications,” in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2013, doi: 10.1007/978-3-642-37057-1_19.
11. P.Andrew, J.Anish Kumar, R.Santhya, Prof.S.Balamurugan, S.Charanyaa, " Certain Investigations on Securing Moving Data Objects" International Journal of Innovative Research in Computer and Communication Engineering, 2(2): 3033-3040, 2014.
12. P.Andrew, J.Anish Kumar, R.Santhya, Prof.S.Balamurugan, S.Charanyaa, " Survey on Approaches Developed for Preserving Privacy of Data Objects" International Advanced Research Journal in Science, Engineering and Technology Vol 1, Issue 2, October 2014