International Journal of Advances in Electronics and Computer Science, ISSN: 2393-2835 Volume-5, Issue-2, Feb.-2018
http://iraj.in

SOFTWARE TESTING TECHNIQUES: A COMPREHENSIVE ANALYSIS

[1]SHLOAK CHOPRA, [2]KSHITIJ MUNDRA, [3]SWARNALATHA P.

[1,2,3]School of Computer Science Engineering, Vellore Institute of Technology, Vellore

Abstract - As our dependence on the software systems is increasing at a breakneck speed- it has become paramount to develop a threshold amount of reliability and trust between us and the software systems. Software Testing is becoming paramount to ensure the promised and expected functional and non-functional requirements are fulfilled by every software system. Software testing has become an inherent part of the Software Development Lifecycle, and has become sacrosanct to test and maintain the sanctity of software. The testing methodologies and techniques should be enhanced and implemented in a proper manner, to ensure that quality is embedded in the software from its conception to deployment. This paper aims to elaborate on existing testing methodologies and discuss upon their application and relations between one another for incorporating better quality assurance. It also elaborates upon some improvements in certain testing methods and techniques.

Keywords - Black-box testing, Grey-box testing, Testing Methodologies, White Box Testing

I. INTRODUCTION

In recent times, our dependence on software system has been growing exponentially and with this growing dependence, the concern for reliability and security in using these software systems is evolving. Software Engineering is of great use in the fields of management systems, system dynamics and computer science [1]. It has become necessary to embed testing techniques at each level in the Software Development Lifecycle i.e. Requirement Specification, Analysis, Design Development, Code Development, Deployment of Software and Maintenance phase. The incorporation of testing methodologies at each phase guarantees the authenticity and quality at that level. The flaws at any particular level can cause serious implications at the cascading levels [2].

The testing process can be divided into various types. When the classification is done based on the stage it is opted for, then it can be divided into- Unit Testing, Integration Testing, Functional Testing, System Testing and Acceptance Testing. While the first four are covered by the software developer or by an engineer who is responsible for quality assurance known as software tester [3], the client or the target users of the software generally do the last. The whole process of testing can also be categorized upon the knowledge of the system undergoing the test; it can be broadly divided into Black Box, Grey Box and White Box Testing. There are various types of testing strategies that are applied with different aim and objective to test a software application.

The Testing Cycle can be broadly categorized into several phases, ranging from the preliminary Test Planning to the final Test Results. The foremost phase is Test Planning and is the review and documentation of all the processes that need to be carried on during the entire testing process. Test Case Development is the next phase in the testing cycle, in which the main aim is the development of test cases, which would be used for testing the software. The third phase of testing is Test Execution where the test cases previously generated are put to use, and the bugs that are relevant are reported in the following phase, which is called the Test Reporting Phase. The final stage of testing cycle is Test Result Analysis where the developer does the defect analysis for the software, and this process can be handled along with the software clients if he has a proper understanding of the development techniques used for development and his requirements [4]. The following terms are usually used in the testing procedures and have been explained in the simplest of means below:

Verification: It is a concept in which it is guaranteed that the software system that has been designed is working in a manner that is acceptable and suitable to the user base.

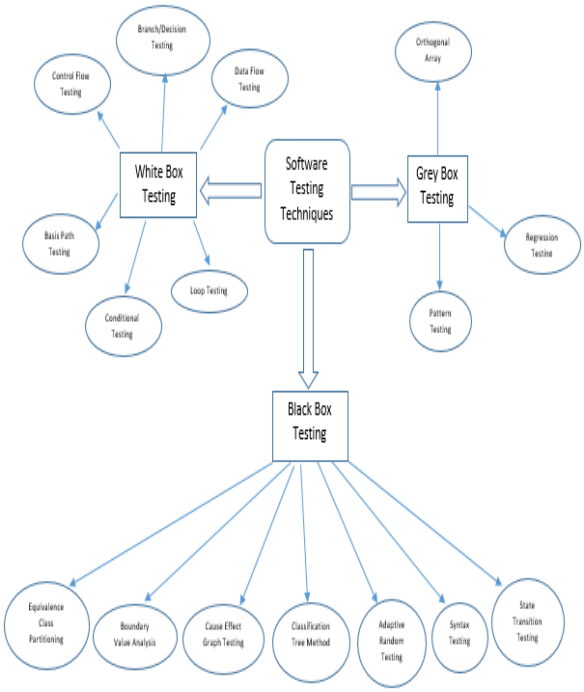

Figure 1: Testing Techniques-Classification

Validation: This is a concept in which it is determined that the software system is developed as it was expected.

II. EXISTING TESTING TECHNIQUES

There are a plethora of existing techniques applied in the domain of Software Testing, but they can be broadly be categorized under the masthead of- White Box Testing, Black Box Testing and Grey Box Testing [5]. The first step is to design proper test cases for each type of testing technique.

White Box Testing Techniques:

The most trivial process of testing the most basic unit of the software- its functionality and the internal architecture of the software system. This methodology of testing requires a certain degree of programming expertise and experience in this domain to design specific test cases. White box testing is also called Security Testing technique and help to certify whether the information system secure the data and perform the corresponding functionality [6]. White box testing techniques are efficient for solving problems as the errors can be detected before they cause any serious implications.

A. Control Flow Testing: The structural or implementation view contains many components, modules or sub-systems that interact with one another. The complex interactions can be subdivided into- interactions that occur along an execution path, in such paths there is a concurrent flow of errors which flows down the entire execution path. The second type are the ones that are specifically related to some data items in execution, here the ones defined latter depend on the earlier ones for values and definitions. Control Flow Graphs (CFGs) are considered a specific case of Finite-State Machines (FSMs) which are used to check the entry, exit, processing and branching nodes for control flow errors.

B. Data Flow Testing: It is one of the oldest testing techniques that has been applied in software testing for more than thirty years now. The data objects are the most basic unit of any software model, and sharing data objects is considered the weakest form of coupling and the strongest form of cohesion in a software system. The primary objective of this method is to derive a set of test cases which execute the paths between the definition and the corresponding use of those data items. The data flow can be tested through Data Flow Graphs (DFGs) which are similar to the CFGs mentioned earlier. The criterion of Data Flow Testing in general requires that definition use associations (duas)should be covered. It aims to achieve that each of the duas should be clearly defined and executed at least once [7].

C. Loop Testing: The loops are generally avoided while evaluating and testing CFGs, because the loop testing methodology is intricately delicate and drastically different. Each time when the loop is traversed, it is counted as a distinct path, but when two or more loops are concatenated, the number of distinct path multiplies, and as the loop is repeated, the combinational complexity exponentially increases [8]. Using the concept of cyclomatic complexity, an often-used software metric we can gauge the logical complexity of a module or modules.

D. Conditional Testing: This type of testing aims at testing all the possible outputs pertaining to any relational or Boolean operator used in any of the modules of the software system. This can be applied in the code development phase of the SDLC, in order to promote quality and reduce the testing time [9].

E. Branch/Decision Testing: Branch testing is closely related to decision testing methodology, and for test items with only one entry point the branch test coverage is equivalent to the decision test coverage. It aims at testing the data flow between modules and produces better results for finite state machines. This applied technique finds a set of paths minimally which contain all primary and internal gate-level input and output lines. It presents a more viable design and test approach. A criteria is used for branch selection, which is called the fitness-function. The two major heuristics used for this fitness function are the approach level and branch level distance. The approach level distance is a more important heuristic in this regard, and helps in increasing the efficiency of branch testing [for details see 10].

F. Basis Path Testing: Every program contains a number of execution paths. This technique aims to analyse only one of those which is the basis or at-least covers most of the intrinsic concepts of execution throughout a program. A basis set is a finite set of linearly independent execution paths, each of which have to be tested at least once. Each independent path has at least one node different from each of all other paths [11]. When tested based on only the logical structure of the program, many infeasible paths are also tested decreasing the efficiency of the testing technique, but when this technique is applied with the control flow graph generated from the code of the software module being tested the efficiency can be improved. On the basis of dependence relationships of the predicates involved a baseline path can be selected instead of the longest path which has many branch nodes, thus increasing the overall efficiency [12].

Black Box Testing Techniques:

The testing approach applied in this type of testing does not include the implementation level detail, as it focuses on the various functionalities that were

expected from the software by the clients, this is collaborated with the help of the requirement specification document. This requires that all the functional requirements should be so stated that they could be tested effectively during the black box testing process.

A. Equivalence Class Partitioning (ECP): The entire input and output domain of the program is divided into classes, and from these classes test cases are derived. Each class in itself represents a set of test cases, which are homogenous amongst themselves, and heterogeneous to those of other classes.

This employs a strategy of 'Divide and Rule' as after division of the entire input domain into equivalence classes. In ECP, it is assumed that if one of the test cases applied works for a partitioned set, then all test cases would satisfy the conditions in relation to that partition set, and if one condition in the partition set fails to work, then we assume that none of the conditions in the partition set would work effectively. The development of equivalence classes requires experience and expertise in the domain of black box testing, as without the proper categorization of equivalence classes, it would be highly unlikely that this methodology of testing would be successful [13].

B. Boundary Value Analysis (BVA): This technique was developed after observing the general trend of an increased amount of errors around the boundary values of various conditional statements within the code. It is based on testing the edge cases-or the boundaries-of the input and output domain. The boundaries can be both valid (within the valid class) and invalid (in the invalid class). It chooses from among the entire domain a set of test cases on the outer-most edges or boundaries of the domain. The following guidelines need to be kept in mind while performing BVA [14]:

(i) The test cases need to be designed in such a manner for any range of values say a and b, such that a and b are considered as the edge cases and would come under boundary value analysis.

(ii) The test cases designed for a set of values should always include the highest and the lowest of the values and the ones above and below them also need to be tested.

BVA is able to identify those bugs, which cannot be directly identified in ECP methodology. In general, both the methodologies are complementary and should go hand in hand during the black-box testing process. There is also another similarity between the two testing techniques that both work at the unit testing level. These are the general parameters to be taken care of while applying these methods.

TABLE V. COMPARISON OF ECP AND BVA

No.	Parameter	Equivalence Class Partitioning	Boundary Value Analysis
1	Number of Test cases	Less	More
2	Quality of testing	Low	High
3	Identified bug count	Low	High
4	Testing time	Low	Slightly High
5	Cost effectiveness	Less	More
6	Testing Effort	Less	More
7	Defect leakage	More	Less

The bug identification rate of the BVA algorithm is higher as compared to that of ECP as it works on the edge cases where there are higher chances of faults. Applying both the techniques complementarily is highly recommended [15].

C. Classification Tree Method: The classification tree methodology focuses on the systematic design and specification of test cases based upon the software system. The application of this testing technique involves two steps- the classification and specification of the input parameters which have to be tested known as the classifications and their parametric values which are in turn known as classes [16]. Classes that result from decomposing the classifications may be partitioned further into sub-classes based upon the quality required during the testing. Each classification acts as a test condition. The process of partitioning applied in this technique is similar to the ECP testing technique, but the stark difference is that the partitions in this methodology should be completely disjoint while in the ECP they could be overlapping based on the problem at hand. Also, the classification tree represents the software system at hand visually.

D. State Transition Testing: This testing methodology derives a model in which the system can be detailed with the help of a specific number of states, transitions between these states and the events driving these transitions. These states should be identifiable, discrete and finite in number. Each event driving the transition should be clearly defined. This model can be represented as a state table or a state transition diagram. The test cases may be either the states or the transitions between them depending on the coverage requirement of testing. As the number of states increase the cyclomatic complexity of the entire model increases, and hence a hierarchical state transition matrix methodology is applied in order to tackle with this complexity [17].

E. Cause Effect Graph Testing: The primary objective of this testing methodology is to derive test-cases that cover the logical relationships between the causes (inputs) and the effects (outputs) of a module or class of the software system. The notation used in

this test methodology is a cause-effect graph. It maps a set of causes to a set of effects assuring to cover all the criterion with about hundred percent efficiency. The cause effect graphs can be effectively derived from the UML diagrams. First the UML diagrams are converted to the decision table, and then it is converted to the cause-effect graph [18].

F. Syntax Testing: It uses a formal model of the inputs to test an item based on the test design. The model is represented as a set of rules, where each of them elaborates upon an input parameter in terms of sequences, iterations or selections. The syntax can be represented in textual or diagrammatic format. The test conditions can be based on either the complete or partial model of inputs. This form of testing gains information from the Software Requirement Specification analysis and source code analysis, and these requirement information are transformed with the help of grammatical parsing to a semantic and syntax based set of rules. It can also help in the automation of the testing approach. The requirement model adds a semantic framework leading to more effective results and better test scripts [19].

G. Adaptive Random Testing: In general testing is considered a time consuming and expensive process, random testing is one of the usual approaches that are used for automatic selection of test cases that need to be tested. The background behind choosing such a technique is to select test cases until a stopping criterion is reached for example detecting a failure, execution of predefined number of test cases or the end of time limitation is met. As the random testing methodology does not benefit from the available information under test, adaptive random testing was introduced. In this method, the random test cases distribution is made with the help of the knowledge of the executed test cases, such that the entire distribution covers the domain of variables and modules to be tested effectively. Distance based or restrictive random testing methodologies are used for adaptive random testing to reduce the overhead in choosing the test cases and to provide an effective coverage over the test domain [20].

Grey Box Testing Techniques:

Grey Box Testing incorporates features from both the black box and white box testing, here the software tester has a basic idea of the structure of the software system and then tests it with respect to the functionalities required. It fuses the concept of structural testing and functional testing [21]. The number of test cases applied are smaller than white box testing approach but larger than black box testing approach.

A. Orthogonal Array Testing: This is a very prominent testing methodology incorporated when the number of possible test cases ranges between a relatively small and an immensely large number. Taguchi further enhanced this method, and his new method is known as Taguchi's Orthogonal Array Testing (TOAT) method. It helps obtain robust solutions and ensure that the incorporated testing scenarios land up to provide a good amount of statistical information with a minimum uncertainty in the operating environment [22]. The orthogonal array testing can be applied at the user interface, system, regression levels and helps identify the faulty logic in the modules or program.

B. Regression Testing: It is the testing methodology that is highly used in dynamic frameworks or where there are incremental rollouts of the software system. It is called regressive because it retests certain functionalities of the existing software whenever additional functionalities have to be integrated into the previous build of the software. It also reuses certain components previously build to increase the efficiency of the entire testing process [23]. This testing method is so developed that the testing framework is designed to integrate into the software development process. This testing technique generates and analyses the performance of the individual components that are added with the new software rollouts or updates, and also tests them with the entire software of the previous build. This testing method is usually used with the Extreme Programming or Incremental SDLC Software Development Models.

C. Pattern Testing: The automatic generation of test patterns with the help of software is an upcoming methodology of testing which is used in domains having projects with similar features. The random test pattern generation does not benefit from the domain knowledge of the internal structure of the program. ATPG methodology aims at reducing the effort of the test engineer without compromising on the test requirements. If the code of the system generated is available at the time of applying this testing methodology, then it can be performed under the hood of white-box testing procedures and if the code is not available by the time of testing it comes under the hood of black-box or rather grey box testing technique. The methodology applied is a mixture of structural techniques in which test suites are generated to cover the structural front, and the functional techniques where random testing techniques are a part of model inference for black-box components [24].

III. APPLICATION OF TESTING METHODOLOGIES

The software testing mechanism is sacrosanct to maintain and test the quality of the software, and hence is applied at different levels in different software applications, expanded below in Figure 3 is

phases at which the quality assurance measures are generally applicable, as observed in the research by Mohammad Kassab [25]. His observations are based on the surveys that have been elaborated in his research. He surveyed 199 participants out of which 167 responded to the survey, and he was able to categorize the following information.

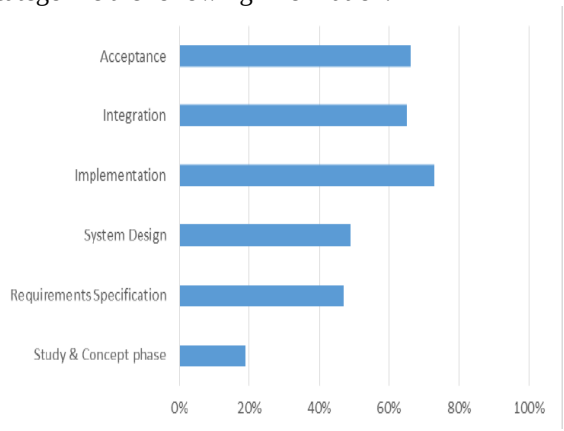

Figure 3-Application of Testing at different levels [25]

CONCLUSION

By and large, testing is one domain that requires expertise and is immensely critical in all the stages of the software development process. The choice of testing methodologies and the level at which they are adapted in the programming is highly important to ensure the timely delivery of software systems. The techniques listed in the paper above are the majority of techniques used across domains of software development. As we note that the extent to which black-box, white-box and grey-box testing techniques are applied depend upon on the software application, but each one of them is applied for a distinct function, and has an important role in testing certain fundamentals of the system.

REFERENCES

[1] Abran, J.W. Moore, P. Bourque, R. Dupuis, and L.L. Tripp, "Guild to the Software Engineering Body Knowledge," IEEE, 2004.
[2] Manish Jain, "Aspect Oriented Programming and Types of Software Testing", Second International Conference on Computer Intelligence & Communication Technology, 2016 pp. 64-69.
[3] Agarwal et al., "Software engineering and testing". Jones & Bartlett Learning, 2010.
[4] Everett et al., "Software testing: testing across the entire software development life cycle". John Wiley & Sons, 2007.
[5] J.Irena. "Software Testing Methods and Techniques", 2008, pp. 30-35.
[6] Muhammad Abid Jamil, Muhammad Arif, Normi Sham Awang Abubakar, Akhlaq Ahmad, "Software Testing Techniques: A Literature Review", 6th International Conference on Communication Technology for The Muslim World, 2016 pp. 177-182.
[7] Roberto Paulo Andrioli de Araujo, Marcos Lordello Chaim, "Data Flow Testing in the Large", IEEE International Conference on Software Testing, Verification and Validation, 2014 pp. 81-90.
[8] Jeff Tian, "Control Flow, Data Dependency and Interaction Testing- Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement" pp. 175-202.
[9] Roger S. Pressman, "Software Engineering-A Practitioner's Approach" pp. 493-494.
[10] Andrea Arcuri, "It Does Matter How You Normalise the Branch Distance in Search Based Software Testing", Third International Conference on Software Testing, Verification and Validation, 2010 pp. 205-214.
[11] T.K. Wijaysiriwardhane, P.G. Wijayarathna, D.D Karunarathna, "An Automated Tool to Generate Test Cases for Performing Basis Path Testing", The International Conference on Advances in ICT for Emerging Regions, September, 2011.
[12] Zhang Zhonglin, Mei Lingxia, "An Improved Method of Acquiring Basis Path for Software Testing, The 5th International Conference on Computer Science & Education, 2010.
[13] Ian Sommeriele, "Software Engineering", Addison Wesley.
[14] T.H. Shivkumar, "Software Testing Techniques", Volume 2, Issue 10, ISSN: 2277 128X.
[15] Pramod Mathew Jacob, Dr. M. Prasanna, "A Comparative Analysis on Black Box Testing Strategies", International Conference on Information Science (ICIS), 2016.
[16] Peter M. Kruse, Joachim Wegener, "Test Sequence Generation from Classification Trees", Fifth International Conference on Software Testing, Verification and Validation, 2012 pp. 540-548.
[17] Kai Cui, Kuanju Zhou, Houbing Song, Mingchu Li, "Automated Software Testing Based on Heirarchical State Transition Matrix for Smart Home", IEEE Transactions, 2017. (Accepted Paper).
[18] Hyun Seung Son; R. Young Chul Kim; Young B. Park, "Test Case Generation from Cause-Effect Graph Based on Model Transformation", International Conference on Information Science & Applications (ICISA), 2014.
[19] Nadia Nahar, Kazi Sakab, "SSTF: A Novel Automated Test Generation Framework using Software Semantics and Syntax", 17th International Conference on Computer and Information Technology (ICCIT), 2014 pp. 69-74.
[20] Korosh Koocheckian Sabor, Stuart Thiel, "Adaptive Random Testing by Static Partitioning", IEEE/ACM 10th International Workshop on Automation of Software Test, 2015 pp. 28-32.
[21] Jai Kaur, Akshita Goyal, Tanupriya Choudhury, Sai Sabitha, "A Walk Through of Software Testing Techniques", 5th International Conference on System Modelling & Advancement in Research Trends, 2016 pp. 103-108.
[22] Han Yi, C.Y. Chung, K.P. Wong, "Robust Transmission Network Expansion Planning Method with Taguchi's Orthogonal Array Testing", IEEE Transactions on Power System, Vol. 26, No. 3, August, 2011 pp. 1573-1580.
[23] Johannes Wienke, Sebastian Wrede, "Continuous Regression Testing for Component Resource Utilization", IEEE International Conference on Simulation, Modelling, and Programming for Autonomous Robots, 2016 pp. 273-280.
[24] Ali Khalili, Massimo Narizzano, Armando Tacchella, Enrico Giunchiglia, "Automatic test-pattern generation for grey-box programs", IEEE/ACM 10th International Workshop on Automation of Software Test, 2015 pp. 33-37.
[25] Mohamad Kassab, Joanna DeFranco, Phillip Laplante, "Software Testing Practices in Industry: The State of the Practice", IEEE Software, Vol. PP, Issue 99, 2016 (Accepted Paper).

★★★