International Journal of Information Technology and Management

Vol. X, Issue No. XV, May-2016, ISSN 2249-4510

AUTOMATED SOFTWARE TESTING: ANALYSIS ON TEST CASE OPTIMIZATION AND TEST CASE LEVELS

AN INTERNATIONALLY INDEXED PEER REVIEWED & REFEREED JOURNAL

www.ignited.in

International Journal of Information Technology and Management
Vol. X, Issue No. XV, May-2016, ISSN 2249-4510

Automated Software Testing: Analysis on Test Case Optimization and Test Case Levels

Ranjeet Kumar[1] Dr. Ramdip Prasad[2]

[1]Research Scholar, Magadh University, Bodhgaya

[2]Associate Prof., Dept. of Mathematics, J. N. L. College, Khagaul, Patna

Abstract – Testing is the method of estimating a system or its element(s) with the committed to find whether it gratifies the quantified necessities or not. Testing is performing a system in order to classify any gaps, errors, or missing necessities in contrary to the actual necessities. It can be definite as - A procedure of examining a software item to identify the modifications among standing and necessary situations (that is defects/errors/bugs) and to estimate the features of the software item (Li, Li, 2009).

Keywords: Testing, Procedure, Software, Life Cycle

----------X----------

1. INTRODUCTION

Testing depends on the procedure and the connected shareholders of the project(s). In the IT industry, large corporations have a team with accountabilities to estimate the established software in framework of the given necessities. Moreover, designers also behavior testing which is called **unit testing**, the following professionals are complicated in testing a system within their respective capacities:

- Software Tester
- Software Developer
- Project Lead/Manager
- End User

Dissimilar corporations have dissimilar descriptions for persons who test the software on the basis of their involvement and knowledge such as Software Tester, Software Quality Assurance Engineer, QA Analyst, etc. (Prather, Jr., 1987).

It is not possible to test the software at any time during its sequence, the next two division's state when testing should be started and when to end it during the SDLC.

Testing is done in dissimilar forms at each phase of SDLC (Prather, Jr., 1987).:

- During the necessity collecting phase, the analysis and confirmation of necessities are also measured as testing.
- Reviewing the project in the design phase with the resolved to increase the design is also measured as testing.
- Testing achieved by a designer on achievement of the code is also considered as testing.

It is problematic to establish when to stop testing, as testing is a never-ending procedure and no one can claim that software is 100% tested. The following features are to be measured for stopping the testing method:

- Testing Deadlines
- Completion of test case execution
- Completion of functional and code reportage to a confident point
- Bug rate falls below a confident level and no high-priority bugs are recognized
- Management decision

2. REVIEW OF LITERATURES:

Software Testing is significant because we all make mistakes. Some of those mistakes are insignificant, but some of them are exclusive or treacherous. We necessity to check everything and everything we produce because things can always go wrong – individuals make mistakes all the time. A common organization for these methods is accessible in

(Arcuri, Yao, 2008, Dunwei, *et al.,* 2011, and B. N. Biswal, *et al.,* 2010). In (Li, Li, 2009) a classification background for automatic test case generation approaches is accessible which is created on software improvement phase in which testing is applied. In (Abdurazik, Offutt, 2000) a classification of search-based automatic test case generation methods is signified there are dissimilar motives to automate test case generation task in software testing. Some of the most significant motives are as follows.

Reducing the cost of software testing - During testing phase the cost can growth more than the probable importance due to incongruous test bags these unfortunate test cases cause wastage of organizational possessions as well as time. There is a necessity to minimalize the cost for receiving a satisfactory product (Object management group, 2003). Reducing human mistakes: In command to find out how a test case is effective there is no confident instrument. It essentially depends on the testers concerned of the necessity. In this method there are lots of human mistakes and tester basic ability level taken into contemplation. This leads to the presence of bugs in the system after testing. To overcome this problematic, automatic test case generation phase should be measured (Prather, Jr., 1987).

Increasing software foodstuffs excellence - It is commonly decided that manual testing is becoming a blockage and is a recurrent cause of project delays particularly for big programs. Therefore, automatic test case project has become significant to certify the excellence of current day large software products (Booch, *et al.,* 2001). Plummeting number of test cases: Generation effective test cases are the necessary permit for streamlining the test work and successful the test effectiveness. The test work is incompetent because of the great number of the primary test cases, so some Computerization processes are necessary to improve the test cases (Deason, *et al*., 1991).

Covering all system necessities - Automating the test case generation procedure provides a means to confirm that test cases have been derived in a dependable and objective method and that all system necessities have been protected (Mingsong, *et al.,* 2006).

3. TEST CASE OPTIMIZATION

Genetic algorithms (Booch, *et al.,* 2001) represent a class of adaptive search methods and measures based on the method of accepted heredities and Darwin's principle of the endurance of the fittest. Genetic algorithm searching mechanism starts with a set of resolution called a population. One resolution in the populace is called a chromosome. The search profits for a number of generations, for each generation the fitter resolutions (based on the capability purpose) will be designated to form a new population. Through the series, there are three main operatives specifically reproduction, boundary and alteration. The cycle will repeat for a number of groups until positive dissolution measures are met. A technique of engendering test bags for operational testing based on genetic algorithms to cover compound goal tracks in one run is obtainable. First, the difficult of producing test cases is expressed as a multi-objective optimization problematic in which the number of purposes reductions beside with generation of test bags. Then, test cases are created using genomic algorithms integrating with dominion knowledge. In (Deason, *et al*., 1991) a Constraint-based Genomic Algorithm method is used to generate improved test cases from UML Activity diagram and Association graphs. (Booch, *et al.,* 2001) Is about test cases produced for software safety based on GA. The technique chooses the most preferred test bags in command to determine the susceptibilities in the software. This technique can decrease the test time and progress the test effectiveness to a positive level. The possibility of collection Pi is definite permitting to the suitability charge fi of the test case di in this algorithm.

Since we assume that our work may have mistaken, hence we all necessity to check our own work. However some mistakes come from bad expectations and blind spots, so we might make the same errors when we check our own work as we made when we did it. So we may not notice the flaws in what we have done.

Ideally, we should get someone else to check our work because another person is more likely to spot the flaws.

There are several motives which obviously tell us as why Software Testing is significant and what are the main effects that we should consider while testing of any product or application.

Software testing is very significant because of the following reasons:

1. Software testing is actually necessary to point out the faults and mistakes that were made during the improvement phases.

2. It's important since it creates sure of the Consumer's dependability and their consummation in the application.

3. It is very significant to confirm the Excellence of the product. Excellence product provided to the consumers helps in gaining their confidence.

4. Testing is required in order to provide the services to the consumers like the distribution of high excellence product or software application which necessitates lower maintenance cost and hence results

into more accurate, dependable and reliable results.

5. Testing is necessary for an effective presentation of software application or product.

6. It's significant to confirm that the application should not result into any failures because it can be very exclusive in the future or in the later platforms of the improvement.

7. It's necessary to stay in the business.

3.1 A COMPARISON OF TESTING METHODS:

The following table lists the points that differentiate black-box testing, grey-box testing, and white-box testing.

Black-Box Testing	Grey-Box Testing	White-Box Testing
The internal workings of an application need not be known.	The tester has limited knowledge of the internal workings of the application.	Tester has full knowledge of the internal workings of the application.
Also known as closed-box testing, data-driven testing, or functional testing.	Also known as translucent testing, as the tester has limited knowledge of the insides of the application.	Also known as clear-box testing, structural testing, or code-based testing.
Performed by end-users and also by testers and developers.	Performed by end-users and also by testers and developers.	Normally done by testers and developers.
Testing is based on external expectations - Internal behavior of the application is unknown.	Testing is done on the basis of high-level database diagrams and data flow diagrams.	Internal workings are fully known and the tester can design test data accordingly.
It is exhaustive and the least time-consuming.	Partly time-consuming and exhaustive.	The most exhaustive and time-consuming type of testing.
Not suited for algorithm testing.	Not suited for algorithm testing.	Suited for algorithm testing.
This can only be done by trial-and-error method.	Data domains and internal boundaries can be tested, if known.	Data domains and internal boundaries can be better tested.

Source from: [9]

4. SOFTWARE TESTING - LEVELS

There are dissimilar stages during the method of testing. In this chapter, a brief explanation is provided about these levels.

Stages of testing include dissimilar approaches that can be used while directing software testing. The main levels of software testing are:

- Functional Testing
- Non-functional Testing

➢ **Functional Testing**

This is a type of black-box testing that is based on the specifications of the software that is to be tested. The application is tested by providing input and then the results are inspected that need to conform to the functionality it was projected for. Functional testing of software is directed on a complete, incorporated system to calculate the system's acquiescence with its quantified necessities.

There are five steps that are involved while testing an application for functionality.

Steps	Description
I	The determination of the functionality that the intended application is meant to perform.
II	The creation of test data based on the specifications of the

	application.
III	The output based on the test data and the specifications of the application.
IV	The writing of test scenarios and the execution of test cases.
V	The comparison of actual and expected results based on the executed test cases.

5. CONCLUSION:

Software testing is the procedure of performing a program in order to find mistakes. Testing is a very significant, though exclusive phase in software improvement and maintenance; it has been assessed that software testing necessitates between 30 percent and 50 percent of software improvement (Li, Li, 2009). A test case is a set of tests achieved in a sequence and associated to a test objective, which will produce a number of tests including definite input values, observed output, estimated output, and any other information necessary for the test to run, such as background requirements (Cunning, Rozenblit, 2005). There has been an important amount of work in automatic test case generation that challenges to growth the amount of detected performance. Despite of these wide researches, there have been few efforts on signifying an all-around organization, which covers all prevailing programmed test case generation methods.

REFERENCES:

A. Abdurazik and J. Offutt, 2000. “Using uml collaboration diagrams for static checking and test generation,” in Proceedings of the third International Conference on the UML. York, UK: Lecture Notes in Computer Science, Springer-Verlag GmbH, pp. 383 – 395.

A. Arcuri and X. Yao, 2008. “Search based software testing of object-oriented containers”, Information Sciences, vol. 178, no. 15, August, pp. 3075-3095.

B. N. Biswal, S. S. Barpanda and D. P. Mohapatra, 2010. International Journal of Computer Applications, vol. 1, Issue 14.

C. Mingsong, Q. Xiaokang, and L. Xuandong, 2006. “Automatic test case generation for UML activity diagrams,” in Proceedings of the 2006 international workshop on Automation of software test, Shanghai, China, pp. 2 – 8.

G. Booch, J. Rumbaugh, and I. Jacobson, 2001. The Unified Modeling Language User Guide. Addison-Wesley.

G. Booch, J. Rumbaugh, and I. Jacobson, 2001. The Unified Modeling Language User Guide. Addison-Wesley.

G. Dunwei, Z. Wanqiu and Z. Yan, 2011. Chinese Journal of Electronics, vol. 19, no. 2.

“Object management group, 2003.” available at http://www.omg.org/uml.

Q. Li and J. Li, 2009. Proceedings of the International Symposium on Intelligent Information Systems and Applications.

Q. Li and J. Li, 2009. Proceedings of the International Symposium on Intelligent Information Systems and Applications.

R. E. Prather and J. P. M. Jr., July 1987. “The path prefix software testing strategy,” IEEE Transactions on Software Engineering, vol. 13, no. 7, pp. 761–766.

S. J. Cunning and J. W. Rozenblit, 2005. “Test scenario generation from a structured requirements specification”, journal of Intelligent and Robotic Systems, vol. 41, no. 2-3, pp. 87-112.

W. H. Deason, D. B. Brown, K. H. Chang, and J. H. C. II, March 1991. “A rule-based software test data generator,” IEEE Transactions on Knowledge and Data Engineering, vol. 3, no. 1,pp. 108 – 117.