IJNRD.ORG ISSN : 2456-4184
INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD) | IJNRD.ORG
An International Open Access, Peer-reviewed, Refereed Journal

AN ANALYSIS OF SOFTWARE TESTING WHITE BOX & BLACK BOX

"HEENA SAINI [1]"

Under The Guidance of:

"MR. SUDHIR KUMAR VERMA [2], DR. NAVIN KUMAR TYAGI [3]"

ABSTRACT

The fundamental objective of software testing is the development of trustworthy, error-free software. It is the goal of software testing to guarantee that the end result satisfies the user's needs. Because of this, software quality and efficiency have seen huge boosts. Software testing is time-consuming and labor-intensive, but it's necessary to make sure the system works as it should. Therefore, it is a tedious and time-consuming task to verify that the system follows the rules. Substituting automation testing for manual testing is one way to overcome the problem of limited testing time. The genetic algorithm is just one of many tactics that have their roots in biological evolution. This approach is based on two pillars: the interpretation and genetic operators. It works well for figuring out if a beginning population is suitable. Because of its importance in genetic algorithms, the fitness function should be strong. Reading up on fitness functions, time, cost, and path coverage revealed that our understanding was weak. To make up for these flaws, scientists have developed a new fitness function using three algorithms and population fitness formulas. The first objective of developing the algorithm was to create a parser capable of counting the instances of iterative and recursive variables within the code. An other method known as GABVIE (Genetic technology based variable importance evaluation) can determine the optimal action by assigning relative values to the code variables. The code demands that the algorithm that has been built accept inputs of different frequencies. The recently constructed fitness function has established the significance of the frequencies of the variables defined by the parser. In order to avoid early convergence and expand the search area, the third technique, 'GABVOCOMO' (Genetic method Based Variation of crossover and mutation operator), uses every possible combination of the genetic algorithm's operators—crossover and mutation. In order to mitigate the total delay brought about by the genetic algorithm's early convergence, this method adjusts to accommodate a diverse population.

Keywords: Software Testing, Genetic Algorithm, test cases, fitness function, path testing, Genetic operators.

INTRODUCTION

Software Testing is crucial to quality and widely deployed by programmers and testers still remains an art, due to limited understanding of the principles of software. The primary purpose of software testing is to develop fault tolerant system with high precision. Software testing ensures that the end product is developed as per the user's requirements. It plays an important role in improving the quality and effectiveness of software. Software testing is an expensive and extensive exercise to rehash the functional requirement of the system and thus it increases the

time complexity and cost of the software. To overcome the time constraint, the manual testing is replaced with automated testing. The concept of the software testing along with the evolutionary algorithm is presented in this chapter.

SOFTWARE TESTING

Today in IT industry, there is a challenge to develop software which is free from error and meets all needs of business. Most of software fails as the bugs are not properly removed before delivery. Software testing is a procedure of finding maximum bugs present in software and produces a defect free and best quality product to the customer (Chauhan 2010).

While testing the software, the applied technique must be operable and observable so that defects can be detected easily and changes in the software can be controlled by the tester. Therefore, for delivering a good quality product, there is need of Software Testing. Software Testing (ST) evolves since the existence of information technology field. Software Testing is a vital phase in Software Development Life Cycle (SDLC). ST is essential to measure the effectiveness of the system product and build trust among the user. Poorly developed software might lead to security ruptures and financial damages. Software testing is very tiring process and consumes more time as compared with software implementation. The literature claims that above 50% of the software development cost is consumed for testing the software. Testing is usually performed for the following purposes:-

✓ To upgrade quality of software: Explorative Testing is usually done to upgrade the quality of the software. That means software is running under all the specified parameters and circumstances.
✓ For the purpose of verification and validation (V & V): Verification and Validation (V & V) are the processes to ensure whether a software system meets all the required specifications or not. It is actually the accountability of software testers to perform verification and validation process which is a vital component of the SDLC
✓ For reliability estimation: Software testing also helps in estimating the reliability. It is used to find the number of bugs, which appears in a predetermined measure of time. In order to estimate the reliability, each module of the program is executed with an intention to find the bugs.

There are three basic steps of Software testing

✓ Creation of test cases for the specific test competence criterion
✓ With the specified test cases execution of software
✓ The computation of the produced output

Test cases that lead to failure are very difficult to identify. Sometimes the system would possibly behave correctly but the faults can also arise over a time period. It is very hard to identify the bugs present in the software with the specified test cases and as a result this will lead the process of software testing more rigid and complex. Therefore by using the above mentioned three steps, the software testing can made be effective. When testing is done successfully with all possible input data only then the developer can claim that the software system is error free. Though the Software testing is not the only criterion that checks the accuracy, there are number of applications that use mathematical analysis for checking the formal verification. But these applications are not scalable. Therefore software testing is essential for finding the bugs that starts from the first step of SDLC.

There are various methods, specifications and parameters on which the software testing should be done. The various stages of software testing are as follows:-

✓ Unit Testing – It is used to test the software system at module level which means all the modules in a system can be tested separately.
✓ Component Testing – It verifies one module or section at a time, that is, verification of each section in an object oriented environment.
✓ Integration Testing – Once all the components have been tested then, through integration of all the components, integration testing has to be done.

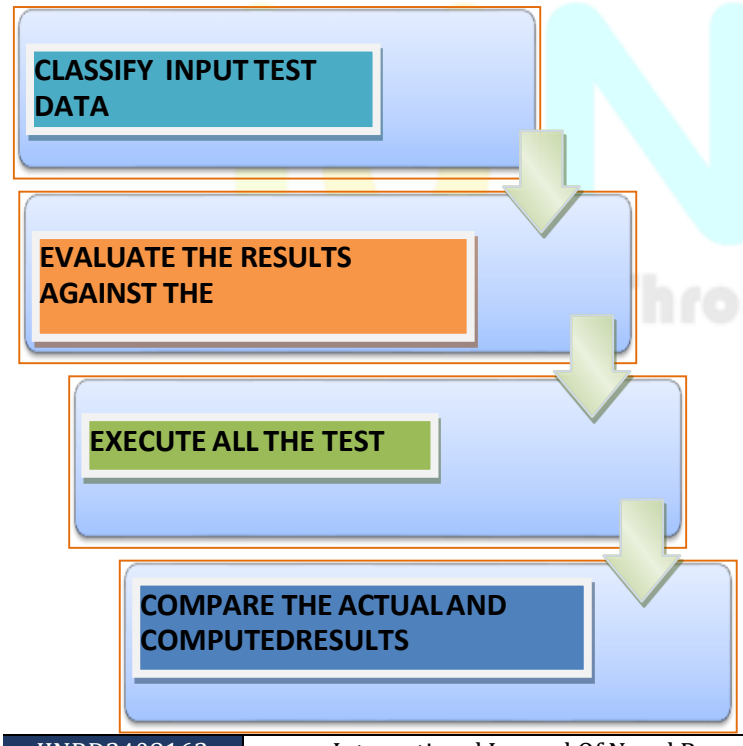
✓ System Testing –The primary aim of System testing is to checks the whole system along with its components.

✓ Acceptance Testing – The primary aim of the acceptance testing is test the software for the acceptability. It is usually performed to determine whether the software met all the requirement specification and is acceptable for delivery

Software testing is classified in to two categories:

Functional (black box) testing (b) Structural (white box) testing

Functional Testing: This is also called the Black box testing. It is called Black box testing as such type of testing does not need any lines of code of the program. This type of software testing validates to identify whether or not the software met all the functional specifications. The motivation behind functional testing is to test the operational behaviour of the software by giving suitable inputs and checks corresponding outputs against the Functional prerequisites. This testing only verifies User Interface, Application Programming Interface, Database, Security, Client/Server communication and other functionality of the software. The primary aim of Functional testing is to verify only the operational behaviour of the software. It stresses on -



✓ Main Functions: Testing the major module of the software.

✓ Applicability: It checks whether a user is free to access the software without any difficulties.

✓ User-friendliness: Checks the system is user friendly

✓ Types of Bugs: It is used to check for the bugs that causes its failure.

The steps of functional testing are as follows:

Steps of Functional Testing (Source: Self)

Since in functional testing there is no need to know the internal coding of the program therefore it totally depends on the input and output of the software. The first step of the functional testing is to create the input test data that check the functionality of the software. There after all the prepared test cases are executed. The output generated after executing the test cases are then compared with actual results to determine the proper functioning of the software.

SOFTWARE TESTING LIFE CYCLE

Software Testing Life Cycle (STLC) means certain steps have to be executed in order to achieve the quality of software. Different phases of STLC process have different deliverables. The testing process in each phase should be accomplished in a systematic manner so that the test cases are designed meaningfully. The following are the different phases of STLC:-

✓ Requirements Analysis- In this phase, requirements gathered in the SDLC process are evaluated for identifying testable aspects

✓ Planning -This phase is related to determine the test planning to identify the resources that helps in achieving the test objective.

✓ Development of test case-This phase involves the creation and reframe of test cases

✓ Test Environment-This phase is very crucial as it determine the condition of both hardware and software under which the testing is to be carried out.

✓ Execution- The test cases based on the test plan are to be executed in this phase.

✓ Closure- This phase evaluate the complete cycle of testing on the basis of time, coverage of testing , Cost and Quality of the software.

Many researchers proposed various algorithms like Evolutionary algorithms, ANN, fuzzy logic, and genetic programming for automatic generation of test data. Any optimization technique whether particle swarm optimization or ANN etc., begin with initial value of the parameters which have been used in the experiments but sometimes these initial values may not be the considered as perfect ones , therefore there is a need to transform these initial values until to get the best one. There are so many ways to generate these initial values, some of them generated randomly and some of them are generated by complex function. The optimization is very important because sometimes the classifier may produce inaccurate results. This inaccuracy is depends on, if the data input is noisy and the bad selection of initial value of parameters. This will affect the overall results of system. There are various optimization techniques available. According to the type of the problem the optimization have categorized in following types.

LITERATURE SURVEY

The combined features of Cuckoo search and genetic algorithm has been used by the authors (Khan, Amjad & Srivastava 2018) for optimization of the test cases. A cuckoo search algorithm first ameliorates arbitrarily generated test cases and after that, genetic algorithm is used to create new test cases. There is a constraint in their algorithm for getting efficient and optimized result. Also the cuckoo search algorithm is applied only for the amelioration of test cases and genetic algorithm is applied for the new generation of test cases, which increase the complexity of the algorithm.

Bashir & Nadeem (2017) develop an algorithm for generating the test cases for mutation testing. Since mutation testing is time consuming and expensive task therefore, the authors have taken genetic algorithm approach for reducing the cost. An object oriented approach has been used to derive the fitness function. The test cases generated were able to cover the statements that comprise mutation. The results were compared with existing standard fitness functions. Only two way cross over and adaptable mutations were used for testing, other versions of crossover and mutation has not been taken into account. The proposed technique was restricted for java codes and not other languages.

Mishra, Mishra, Acharya & Das (2019) have proposed a hybrid method using genetic algorithm approach suitable for mutation testing and path testing. In the proposed algorithm the authors generated the test data that cover paths and detect all the mutants inserted in the program code. Also Fault determination table is used, that stores all the test data used to cover the mutants and delete the duplicate data if same mutants covered again. A single point cross over and bitwise mutation has been covered in the proposed algorithm. The results were obtained by testing only unit module of iterative program code.

The generation of automatic test cases have been implemented by the authors (Sharma, Rishon & Aggarwal 2016) using genetic algorithm for path testing. Test cases were implemented through three different tools i.e. Matlab, Ruby and C++ for unit testing and the results were compared with standard random testing. But the proposed algorithm was only implemented on iterative code. The test cases generated were not tested for recursive program.

Khan & Amjad (2015) have generated automatic test cases using Genetic Algorithm in order to increase the efficiency of software testing. In this paper, the author had suggested that random method was not appropriate method for the selection process of test data. The authors have used mutation analysis and genetic algorithm approach for producing test data for program code. The procedural control flow graph (CFG) was used for the program. The weight of CFG was distributed among all edges according to the pareto principle. However the algorithm introduced was

suitable for unit testing but in order to test the overall code, mutation testing is not effective.

METHODOLOGY

The new parser is designed to compute the frequency of variables which are executed throughout the program that has been used in an algorithm to find optimal path. The parser is implemented in python language. The IDE used for its implementation is Jupyter.

The new fitness function is designed using mathematical formula (derived from neural network) for evaluating the fitness of the population. The mathematical formula works on the binary string.

A new designed algorithm namely 'GABVIE' (Genetic Algorithm based variable Importance evaluation) used for identifying the feasible optimal path of the program code for generating the test cases. The new algorithm is also implemented in python language. The IDE used for its implementation is Jupyter.

An extended algorithm namely 'GAVCOMO' (Genetic Algorithm based variation of cross over and mutation operator) works on variation of crossover and mutation operator. It is implemented in Python language. The IDE used for its implementation is Jupyter.

The results obtained by the new algorithms are compared with the existing algorithms. The obtained results have been tested on various iterative and recursive programs based on 'C' language.

CHALLENGES

Attempts have been made in the literature to deal with the shortcomings of Software Testing. Some of the works are discussed below to overcome the shortcomings like coverage, time and quality in software testing.

The authors (Yao, Gong, Li, Dang & Zhang 2020) have generated the algorithm for testing the program with the random values. They have built the mathematical model for the optimization by using a set-based genetic algorithm. The algorithm focused only one target at a time that means either to cover branch, statement, condition or path. However the proposed algorithm was restricted to solve the program with random numbers

Panichella, Kifetew & Tonella (2017) have developed Dynamic Many-Objective Sorting Algorithm (DynaMOSA), which was a multiple - objective solver. It was specially designed to address the test case generation problem in the context of coverage testing. The experimented have been conducted on 346 java programs with the different data sets. The objective of the Algorithm was to cover branch and mutants applied in the program. Other parameters like memory consumption, execution time, number of generation and number of paths have not taken into account.

The authors (Yao, Gong & Wang 2015) have developed the algorithm for improving the performance of test data generation for multi-path coverage. The mathematical model has been established for all target paths. In order to cover multiple paths the author used multiple fitness function that increases the complexity.

RESEARCH GAPS

The literature discussed above provides the various methods for path testing using genetic algorithm. There are still many gaps which are as follows:-

In the existing algorithms, the researchers used the control flow graph for generating the test cases in path testing. They have not used the variables as a constraint of the program for obtaining the maximum path. Also, no parser is available for determining the frequency of the variables defined in the program.

The existing fitness functions applied by the various researchers lead to the inaccurate results if the size of the initial population is not constant.

The existing algorithms generate the test cases for both feasible and infeasible paths as they scan

entire path coverage of program code again and again which increases the time and space complexity.

Since there are multiple variations of crossover and mutation operator for generating the test cases in path testing but the existing algorithms have generated the appropriate test cases using only single point crossover and one- bit mutation operators. This process reduces the accuracy of generating test cases and thus gives non-optimal solution.

The above reasons motivated the author to work on white box testing to identify optimal path for test case generation.

RESEARCH OBJECTIVES

In view of the limitations of the existing algorithm, the objectives of the study are:

✓ To design a parser for evaluating the frequency of the variables defined in both iterative and recursive program.
✓ To develop the new fitness function for finding the fitness of population
✓ To design an algorithm for finding the optimal path for test case generation in path testing with reduced time and space complexity.
✓ To design the algorithm for generating optimal path by using multiple variations of crossover and mutation operators for enhancing the search space.

SOFTWARE AND HARDWARE USED

Software Testing Life Cycle (STLC) means certain steps have to be executed in order to achieve the quality of software. Different phases of STLC process have different deliverables. The testing process in each phase should be accomplished in a systematic manner so that the test cases are designed meaningfully. The following are the different phases of STLC:

✓ Requirements Analysis- In this phase, requirements gathered in the SDLC process are evaluated for identifying testable aspects
✓ Planning -This phase is related to determine the test planning to identify the resources that helps in achieving the test objective.
✓ Development of test case-This phase involves the creation and reframe of test cases
✓ Test Environment-This phase is very crucial as it determine the condition of both hardware and software under which the testing is to be carried out.
✓ Execution- The test cases based on the test plan are to be executed in this phase.
✓ Closure- This phase evaluate the complete cycle of testing on the basis of time, coverage of testing , Cost and Quality of the software.

Many researchers proposed various algorithms like Evolutionary algorithms, ANN, fuzzy logic, and genetic programming for automatic generation of test data. Any optimization technique whether particle swarm optimization or ANN etc., begin with initial value of the parameters which have been used in the experiments but sometimes these initial values may not be the considered as perfect ones , therefore there is a need to transform these initial values until to get the best one. There are so many ways to generate these initial values, some of them generated randomly and some of them are generated by complex function. The optimization is very important because sometimes the classifier may produce inaccurate results. This inaccuracy is depends on, if the data input is noisy and the bad selection of initial value of parameters. This will affect the overall results of system. There are various optimization techniques available. According to the type of the problem the optimization have categorized in following types:

a) Constraint and unconstrained optimization

b) Discrete Optimization and Continuous Optimization

c) Single and Multi - objective Optimization

d) Multimodal Optimization

e) Combinatorial Optimization

Constraint and unconstrained **optimization:** When the constraints are imposed on the variables or the given problem to solve it is called constraint optimization. Constraint means that some limitations are imposed on the variables that prevent them from going in specific direction. It is mostly applied in nonlinear programming where the optimum values can occur at the boundaries. In unconstrained optimization, no constraints are specified on the variables or the problem. That means either no boundaries are given or if the boundaries are given then it must be softer.

Discrete Optimization and Continuous Optimization: Problem optimization works on discrete set of variables are called discrete optimization. If the value of the variables can be any number from the set of real numbers and there are no gaps between the numbers, then it called continuous optimization. The continuous optimization is easy to use as compared to discrete optimization. The continuous optimization is used when there is requirement of continuous variables in objective functions to solve the problem more efficiently.

Single and Multi-objective optimization: When only one target point is stated in optimization problem then it is called Single objective optimization. Feasibility problem is an example of single optimization as the objective is to find the value of the variables that satisfy the constraint. On the other hand when optimization problem comprises more than one target function which has been optimized simultaneously, it is defined as multi-objective optimization. There are many fields where the multi-objective optimization problems occur like economics, engineering, mathematics and many more.

Multi-modal Optimization: Multi-model optimization deals with finding various optimal solutions (global or local) so that the user can have better understanding of the all optimal solution. It is used to optimize the multi model function with local optimum solution. As per the requirement, user can also switch to the other optimal solution.

Combinatorial Optimization: The main purpose of the combinatorial optimization is to find the best possible solution from the set of the finite solutions. This set enumerated very large set. It is also used to find the efficient algorithm for computing the best feasible optimal solution. It is also used to evaluate the combinatorial structures (like graphs structure, matching problems etc.), creating the connection between various combinatorial optimization, examine the complex problems and to compute its complexity.

CONCLUSIONS AND FUTURE WORK

By developing novel algorithms that can produce optimal paths according to the frequency of variables specified in the program's code, the work reported in the thesis contributes to the current body of information. The frequency variables are output by the recently built parser. A fitness function is built to ascertain the most significant variables. A new fitness function, the program's feasible optimal path, an evaluation of the optimal solution, and variable frequencies can all be calculated by the algorithms that have recently been developed using different combinations of mutation and crossover operators. The findings presented in the work were derived from a thorough empirical analysis. The results generated by these newly-developed algorithms are clearly more accurate than those of the original algorithms and their variants. Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Bee Colony Optimization (BCO) can all make use of the newly derived fitness function. The initial new algorithm (parser) was created to find the frequency of use of the program's defined and executed variables. In both cases, it can help you find the best course of action. Better software testing is just one of many potential applications; it will also help with optimizing network routing paths, utilizing fuzzy logic (an A* search algorithm), and solving problems like the Traveling Salesman Problem. Similarly, the new algorithm (GAVCOMO) that changes the mutation and crossover operator improves the search space. It can be used to

increase the size of the search spaces for both exploration and exploitation.

FUTURE WORK

This study's overarching objective is to identify the optimal strategy for traversing source code as quickly as feasible. The GABVIE algorithm has many applications; for instance, it is used by artificial neural networks (ANNs) to create fitness functions that can return values between 0 and 1, by network routing algorithms to solve the shortest path problem (TSP), by fuzzy logic to recognize patterns, and many more. In this study, a fitness function is employed to evaluate the fitness of chromosomes and to ensure that the next generation inherits the most fit variants. Additionally, the extended fitness function can ascertain the fitness for impractical paths. A roulette wheel selection procedure is used in the research to find the best candidates. Using additional selection techniques like tournament selection and stochastic selection can further simplify optimization and searching in future work. The proposed GABVIE algorithm can be improved by incorporating quantum and diploid genetic algorithms.

REFERENCES

1. Ahmed, A., H. & Taha, D., B., 2018, 'Generation of Optimal Testing Paths using Anti–Ant Colony Algorithm', International Journal of Computer Applications 179(1), 20-25.
2. Ahmed, M., A. & Hermadi, I., 2008, 'GA-based multiple paths test data generator', Computers & Operations Research 35(10), 3107-3124.
3. Alba, E. & Chicano, F., 2008, 'Observations in using parallel and sequential evolutionary algorithms for automatic software testing', Computers & Operations Research 35(10), 3161-3183.
4. Alzabidi, M., Kumar, A. & Shaligram, A.,D., 2009, 'Automatic Software structural testing by using Evolutionary Algorithms for test data generations', International Journal of Computer Science and Network Security 9(4), 390-395.
5. Baresel, A., Sthamer, H. & Schmidt, M., 2002, 'Fitness function design to improve evolutionary structural testing', In Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation , London, July 2002, pp. 1329-1336.
6. Bashir, M., B. & Nadeem, A., 2017, 'Improved genetic algorithm to reduce mutation testing cost', IEEE Access 5(1), 3657-3674.
7. Bashir, M., B. & Nadeem, A., 2013, 'A fitness function for evolutionary mutation testing of object-oriented programs', In 2013 IEEE 9th International Conference on Emerging Technologies (ICET), Islamabad, December 2013, pp. 1-6.
8. Bhasin, H. & Bhatia, S., 2011, 'Application of genetic algorithms in machine learning', International Journal of Computer Science and Information Technologies (IJCSIT) 2(5), 2412-2415.
9. Biswas, S., 2018, 'Proposal for Control Dependency White-Box Test Coverage Metrics for Inheritance', In 2018 2nd International Conference on Data Science and Business Analytics (ICDSBA), Changsha, September 2018, pp. 155-160.
10. Blanco, R., Tuya, J. & Adenso-Díaz, B., 2009, 'Automated test data generation using a scatter search approach', Information and Software Technology 51(4), 708-720.
11. Boopathi, M., Sujatha, R., Kumar, C.,S. & Narasimman, S., 2014, 'The mathematics of software testing using genetic algorithm', Proceedings of 3rd International Conference on Reliability, Infocom Technologies and Optimization, Noida, October 2014, pp. 1-6.
12. Bouchachia, A., 2007, September, 'An immune genetic algorithm for software test data generation', In 7th International Conference on Hybrid Intelligent Systems (HIS 2007), Kaiserlautern, September 2007, pp. 84-89.
13. Bueno, P.,M.,S. & Jino, M., 2002, 'Automatic test data generation for program paths using genetic algorithms', International Journal of Software

Engineering and Knowledge Engineering 12(06), 691-709.
14. Bühler, O. & Wegener, J., 2008, 'Evolutionary functional testing', Computers & Operations Research 35(10), 3144-3160.
15. Cao, Y., Hu, C. & Li, L., 2009, 'Search-based multi-paths test data generation for structure-oriented testing', In Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation, Shanghai, China , June 2009, pp. 25-32.
16. Chauhan, N., 2010, 'Software Testing: Principles and Practices', Oxford university press, India.
17. Chen, Y., Zhong, Y., Shi, T. & Liu, J., 2009, 'Comparison of two fitness functions for GA-based path-oriented test data generation', Proceedings of Fifth International Conference on Natural Computation, Tianjian, China, January 2009 4(1), pp. 177- 181.
18. Clarke, L., A., 1976, 'System to generate test data and symbolically execute programs', IEEE Transactions Software Engineering 3(1), 215-222.
19. Črepinšek, M., Liu, S.H. & Mernik, M., 2013, 'Exploration and exploitation in evolutionary algorithms: A survey', ACM computing surveys (CSUR) 45(3), pp. 1-33.