Minimizing The Size Of Test Suite Using Genetic Algorithm For Object Oriented Program

Shubham Kothari
Research Scholar, Computer Science Department
Shri Vaishnav Institute of Technology & Science
Indore (M.P.), India
shubhever.cs@gmail.com

Anand Rajavat
Head, Computer Science Department
Shri Vaishnav Institute of Technology & Science
Indore (M.P.), India
anandrajavat@yahoo.co.in

***Abstract*— Software development and management of large scale projects are the complicated task. To support the software developers, the object oriented methodologies are used for reducing development efforts. But still a significant amount of efforts and team work is required to design and develop the systems according to the users need. Among them the testing is a one of the key components of software development life cycle. The testing assures about the quality of software development and their services. In this presented work, a new software testing model is proposed and implemented that offers the automated testing with test case generation, optimization and code evaluation. The implementation of the future method is provided using the JAVA based technologies and their performance is also computed with number of different experiments. During experimentations, either the LOC (line of code) increases to measure the stability of performance or optimization steps increase to find the efficiency of the proposed model. The experimental results demonstrate that the future method is capable to analyse small as well as large scale projects with the less resource consumption and more accurately.**

Keywords—Software engineering; SDLC; testing; test set optimization; implementation; performance assessment

I. INTRODUCTION

Now in these days, most of the organizations and individuals are preferred to work in digital environment. In different manners, such as email messages, online banking, inventory systems and others are the key areas of digital data storage and utilization. In order to manage the data, we need to develop some software and application. The application or software development is a critical process which involves understanding, design, negotiation, finalization and other tasks. In other words the entire process of software development needs a significant amount of efforts. The made efforts for an efficient and effective software development can be summarized as SDLC (software development life cycle) [1].

According to different software engineers and developers the SDLC models can be of different kinds or has less and more steps to perform software development task. But in most common format, the SDLC model has requirement gathering, analysis, design, coding, testing and deployment [2]. The process stack is given using figure 1.1. In this context the word "stack" shows the impact of one phase into another. In other words, the involve processes depend on one another. Therefore

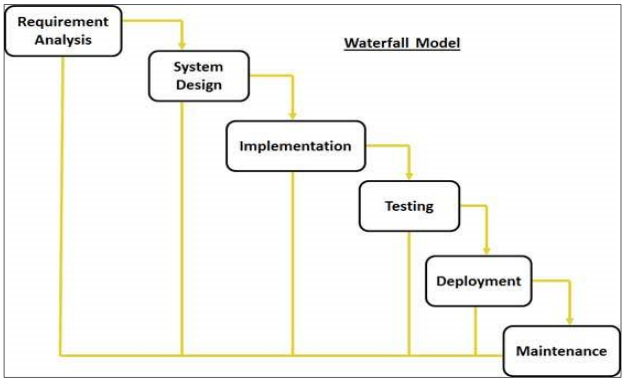

Fig. 1.1 SDLC process

if first phase is completed successfully then the next phase can be performed in easier manner.

In first step, the negotiation with client is performed for understanding of actual needs or requirements. In further the requirements are analysed to find the feasibility and then the design is prepared. After designing of the system, the code is written for implementation of the system and then testing is performed. Testing helps to evaluate implemented code blocks with respect to the client requirements. Therefore the testing is the key of quality software development because during testing the code blocks are tested to find bugs and the issues in functional execution of the system. If the entire developed module is fit or bug free then it is deployed at the client end. Therefore in other terms, the testing assures developers for the quality in software development [3].

In this presented work, the testing of the software system [4] is the key area of interest. The key issue in testing is to prepare a set of test cases, employ on the functions and getting the outcomes. This process is a time consuming process, by which the cost of the software increases significantly. In order to reduce the efforts and cost of the testing or the software quality assessment, a number of research articles suggests the optimization of the test cases. The optimized test cases are used with different functions to cover most of the code blocks. Thus using this hypothesis a new model for automated testing is prepared using the genetic algorithm. This section provides the overview of the proposed work. In next section, the genetic algorithm and their basics are provided

.

978-1-5090-5515-9/16/$31.00 ©2016 IEEE

paperdownloader.com
دانلود رایگان مقاله

II. GENETIC ALGORITHM

Genetic algorithm [5] is one of the most admired search techniques which is used in various different applications to solve different issues of real world problems. Some of the applications are available in the domain of optimization, search, decision making and others. The genetic algorithm consist of four main phases of data analysis or search, namely selection, cross over, mutation and updating [6]. Each module includes the process to find the best solution among the available solutions. The simple steps of the processes involved in the genetic algorithm are shown using figure 2.1.

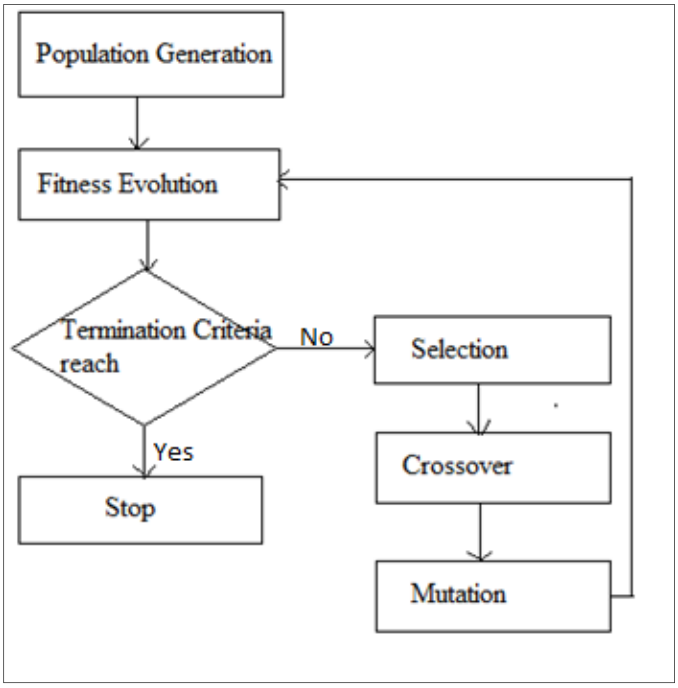

Fig. 2.1 Genetic algorithm

In population generation phase, the possible symbols in the solution sequences are used to generate the population. Thus to generate the different combination of solutions the random process is used with the symbols. After that each randomly generated population is tested with the fitness function. The fitness function is a kind of heuristic which measures the generated population whether the population sequences are providing solution or not. Additionally if that provides the solution then "how much satisfactory solution is?". After finding fitness of individual solution, the solution needs to optimize more. So, the termination criterion is evaluated first. In genetic algorithm, two key termination conditions are frequently used.

1) The maximum number of evaluation cycles are completed

2) Or, the required solution is obtained

In this context, if any one condition is satisfied then the process is terminated otherwise the selection process is used. To select the solutions from the generated population sequences again a random process is used to select the solutions from the initial population. But, the issue is that how many solutions are selected for further process. Therefore the following formula is used to find the amount of solutions required to select [7].

$$ne = \alpha * \beta$$

Where ne= number of elitism

α = number of population

β =rate of elitism

In this step the best ne solutions are selected for next process. Now in next step, crossover operator is used for generation of new solutions using the selected solutions. Thus at a time two solutions are selected and the cross over is performed among both of them. In a single point cross over, both the solution sequences are divided into two parts head and tail, additionally the interchange of heads or tails are performed in this operation. In next phase the mutation is performed on the newly generated solutions. Means the outcome of crossover is used with mutation. The mutation is responsible to make the validation whether "the solution is feasible or not?". If the generated solution is not a feasible solution then the repairing is performed in this step, Finally, if the solution is repaired then it is updated on the population otherwise the solution is discarded. Again the entire process is performed until the termination condition is reached.

In this section the basics of the genetic algorithm for optimizing the solution is reported. In next section the proposed work is demonstrated.

III. PROPOSED WORK

The key aim of the proposed work is to optimize the test suite [8] for maximizing the coverage of the test cases to test the code blocks. Additionally to reduce or minimize the time consumption of the test case writing and execution for the object oriented language. In existing system, six C Programs were used as subjects [11] but in proposed system we deals with object oriented approach. Therefore the JAVA programming technology is selected for work. Therefore a new model using existing tools and concepts are prepared. The design of the proposed technique is performed in two major modules- the test case generation and the minimization or optimization of generated test cases. Both of the scenarios are defined as:

A. *Test case generation*

In this phase, the test cases are generated for performing functional test on the input code blocks. In this phase, initially the system accepts the code files which are required to be tested. The system analyse the entire code by parsing them into the classes, methods, constructors and the other variables. These extracted data from the input code files are stored temporarily. Further the encoding is performed for the generated set of code elements. By which the binary sequences are prepared for further use.

After encoding of data, the genetic algorithm starts working on the test case generation modules. Thus the random population using the binary symbols is prepared. The generated random population is tested for finding the fitness values of the generated test cases. In order to test the fitness of the test cases JUnit [9] and EMMA tool [10] is used. After fitness evaluation; the selection, crossover and mutation is performed to find the best fit test cases. The entire process is summarized using the following algorithm steps.

1) Input JAVA source code to system

2) Input code is analysed to perform the following operation on CUT (class under test)

a) Parse the source code and extract the type information from source code

b) Extract the constructor, methods and functions with their signatures and impact

c) Store the extracted information into a data structure

3) Initialize the population with the two symbols 0 and 1 in the random manner. The colony size of the population is

similar to the length of extracted information or prepared data structure

4) In each individual solution sequence 1 denotes particular method considered for testing and 0 shows method not considered for testing.

5) The fitness of each generated solution is calculated in following manner:

a) First the encoded solution is decoded into constructors and methods.

b) Now test case statements are added with two parameters- expected value and value generated by the method.

c) The test case is executed using JUnit tool that return whether the test case approved or failed.

d) If test case is passed successfully through JUnit then coverage of test case over the code blocks is calculated through EMMA tool. The coverage value is used as fitness of the solution.

6) If the termination condition arrived then stop the algorithm. Else after that production is calculated by using assortment, mutation and intersection and again go to step 5.

7) The final population shows the test cases. Now, combine all the test cases to make a test suite and calculate the fitness of the test suite.

The entire process of test case generation is defined in this section. Additionally the flow diagram of the test case generation technique is given using figure 3.1. In next section the optimization of generated test cases is provided.

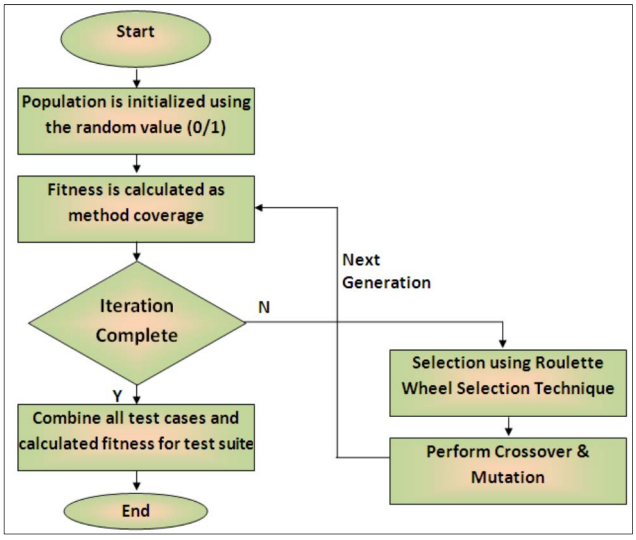

Fig.3.1 Test case generation

B. *Test case optimization*

This section provides the details that how the generated test cases are optimized to minimize the testing efforts in terms of test case coverage and the required time to execute the test cases. To optimize the test cases again the genetic algorithm is used to search most fit solution among the generated test cases. Below given algorithm steps demonstrate how the test cases are optimized. The flow diagram of the optimization of generated test cases is given using figure 3.2.

The proposed steps for test case optimization for reducing the size of test suite are as follows:

1) As similar to the previous module, the population are generated randomly with the help of symbols 0 and 1.

2) Compute the fitness value for each generated resolution using the following steps:

a) The generated solutions are decoded to test cases according to corresponding method of the test case called.

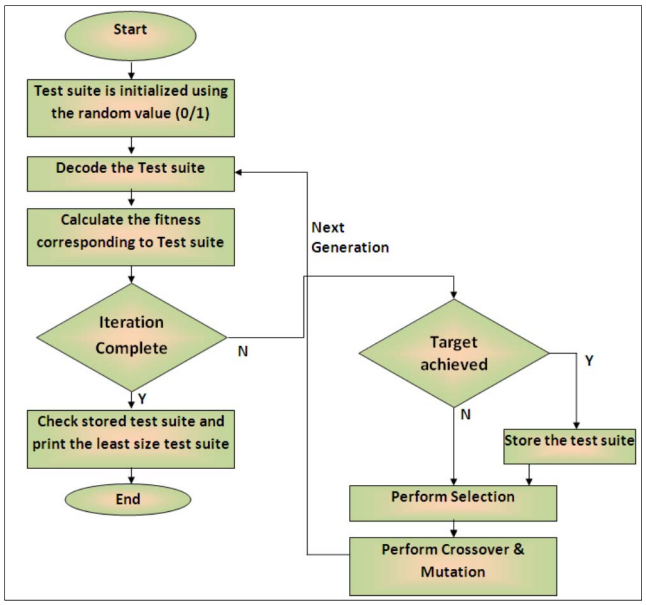

Fig. 3.2 Optimization of test case

b) The fitness of the whole test suite using JUnit and EMMA tool.

3) Sort the population according to fitness criteria and find the solution that have high fitness value i.e. coverage.

4) If test suite having fitness value greater or equal to fitness value of the entire test suite then store this solution sequence separately.

5) Check for termination conditions, if termination condition appears then check the separated test suite. Else the next production is calculated by using assortment, mutation and crossover. Again go to step 3.

6) In the stored test suite, print the test suite having least number of test cases as the result.

IV. RESULTS ANALYSIS

After implementation of the proposed test case optimization method, the system computed different parameters. The obtained experimental results of different parameters are provided in this section.

A. *Time consumption*

The amount of time required to generate and optimize the test cases are reported here as the time expenditure of the proposed system. The figure 4.1 shows the time consumption of both the modules of test case automation and optimization.

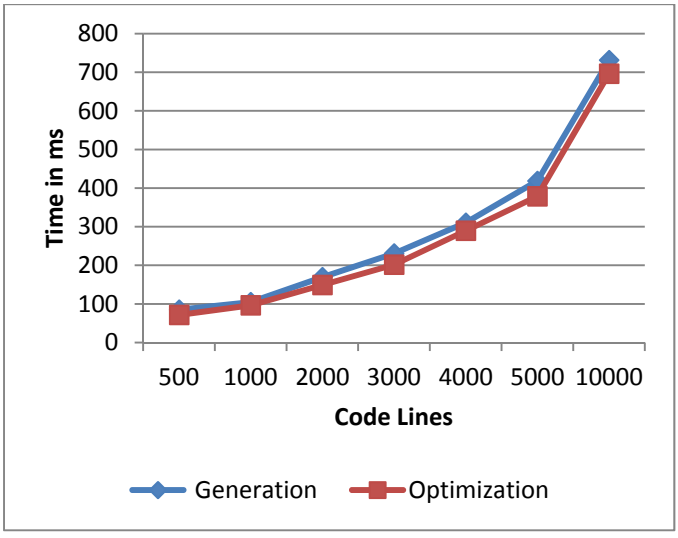

Fig. 4.1 Time consumption

In order to demonstrate the performance of both the modules blue line is used to demonstrate the time consumption during generation of test cases and the red line indicates the time consumption during optimization. The X axis contains the amount of code lines produced for evaluation and Y axis shows the required time for test case generation and optimization. The measured time is reported here in milliseconds. According to the obtained performance both the techniques require adoptable time for test automation. But according to the results the test case generation needs higher time as compared to optimization.

B. *Memory requirements*

The amount of main memory for processing the input code to generate and optimize the test cases is reported as the memory consumption. The obtained memory consumption is reported using figure 4.2.

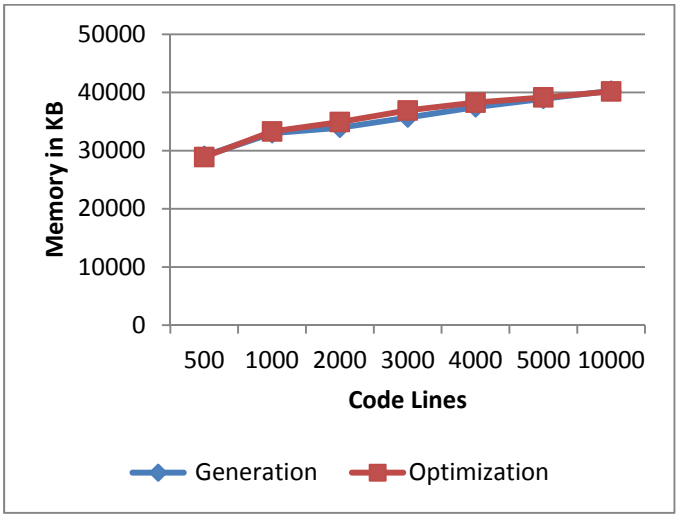

Fig. 4.2 Memory consumption

The amount of main memory consumption is reported for both test cases generation and optimization. The blue line shows the presentation of generation phase and the red line shows the performance of optimization phase. In this diagram the X axis shows the input file size in terms of LOC (line of code) and the Y axis shows the amount of main memory in terms of KB (kilobytes). According to the results, both the processes require similar amount of main memory for execution.

C. *Code coverage*

The coverage is also used in the test case automation as the fitness function for evaluating the test cases. But the means of code coverage is little bit change here for performance approximation.

The amount of LOC (line of code) covered after optimization of test cases are termed here as the code coverage. Thus for the demonstration of the covered codes by optimizing the test cases are given using the figure 4.3. To evaluate the coverage of code lines, the EMMA tool is used and the mean percentage is computed to denote the entire LOC. In this diagram the X axis contains the input LOC and the Y axis provides the coverage of code in terms of mean percentage. According to the obtained performance, the proposed test case optimization system provides the optimal code coverage in different number of experiments.

D. *Optimization*

This parameter indicates that after how many number of iterations the system becomes able to produce the optimal solution. Therefore with the 1000 LOC (line of code) the different numbers of experiments are performed and their code

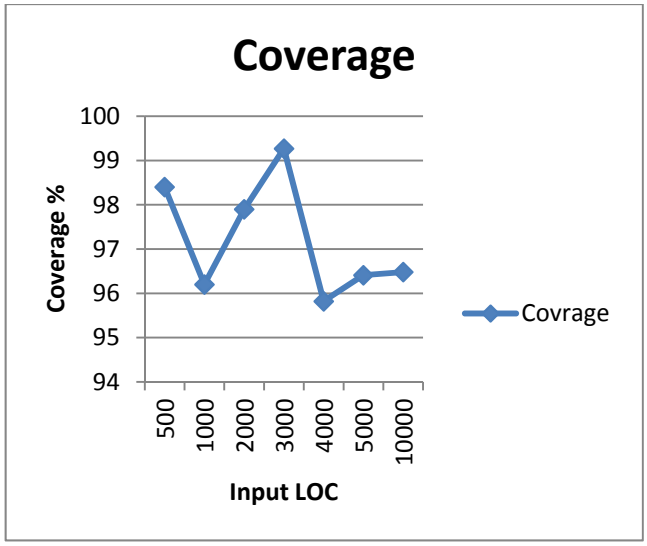

Fig. 4.3 Code coverage

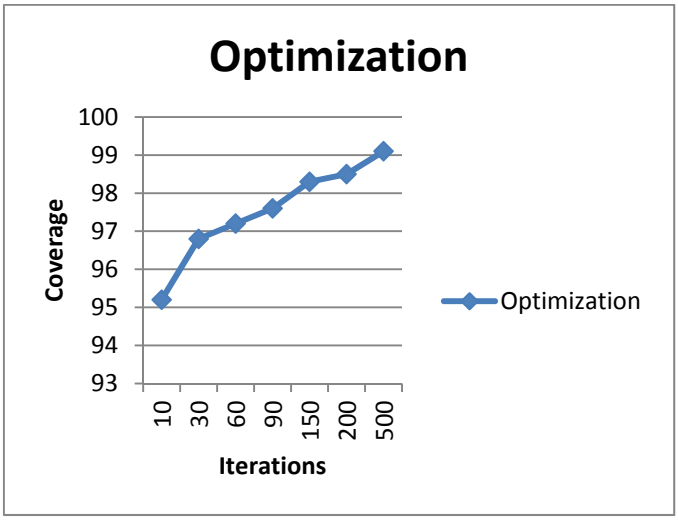

Fig. 4.4 Optimization

coverage is estimated. The figure 4.4 shows the evaluated performance of the algorithm with the increasing number of iterations.

The figure 4.4 shows the code coverage of the algorithm with the increasing number of iterations. In order to show the performance of the proposed technique X axis contains the number of iterations and the Y axis shows the code coverage of the algorithm in terms of percentage. According to the obtained performance initially the coverage is rapidly growing and after that it is less frequently growing. Thus the future method is capable to optimize the code coverage issues more effectively as required.

V. Conclusions

In the software development life cycle, the testing is essential part of product or application development. The testing of the product ensures the end user to work reliably with the system for long time. In this presented work, the software life cycle is studied in detail and it is decided to work for software testing. In literature a number of automated testing tools are available. These tools promise to perform the testing automatically with large code coverage. But in most of the techniques, the automated testing is performed effectively but the coverage issue remains unchanged. Therefore using the genetic algorithm a new method is essential that not only perform automatic testing but also helps to minimize the resource consumption by minimizing the efforts for testing and test case generation for large amount of code lines. This

application is developed with the help of JAVA technology and the JUnit testing tools. After implementation of the required concept, the performance of the implemented system is also computed in different performance parameters. The obtained performance is summarized in the following table I.

TABLE I. Performance Summary

S. No.	Parameters	Remark
1	Time complexity	Less time consuming in processing large scale projects too
2	Space complexity	Less space complexity for small and large application testing
3	Code coverage	Enhancing the code coverage with increase in amount of LOC
4	Optimization	Rapidly changing with small optimization steps

According to the presentation of the proposed system, the model is efficient and optimizes the test cases to improve the performance of the system in terms of time and space complexity.

REFERENCES

[1] Dr. H.S.Behera, Asst. Prof K. K. Sahu, Asst. Prof Gargi Bhattacharjee, "LECTURE NOTES ON SOFTWARE ENGINEERING", DEPT OF CSE & IT VSSUT, Burla, Course Code: BCS-306

[2] "Systems Development Lifecycle: Objectives and Requirements is", Bender RBT Inc. 17 Cardinale Lane Queensbury, NY 12804 518-743-8755

[3] ALAN DENNIS, BARBARA HALEY WIXOM, ROBERTA M. ROTH, "SYSTEM ANALYSIS AND DESIGN", John Wiley & Sons, Inc.

[4] Bernd Bruegge, "System Testing: Software Engineering 1: Lecture 16", Applied Software Engineering Technische Universitaet Muenchen

[5] Ulrich Bodenhofer, "Genetic Algorithms: Theory and Applications", Lecture Notes Third Edition—Winter 2003/2004

[6] MATEJ CREPIN SEK, SHIH-HSI LIU, MARJAN MERNIK, "Exploration and Exploitation in Evolutionary Algorithms: A Survey", ACM Computing Surveys, Vol. 45, No. 3, Article 35, Publication date: June 2013.

[7] Jifeng Xuan, He Jiang, Zhilei Ren, "Pseudo Code of Genetic Algorithm and Multi-Start Strategy Based Simulated Annealing Algorithm for Large Scale Next Release Problem", Dalian University of Technology

[8] Aftab Ali Haider, Shahzad Rafiq, Aamir Nadeem, "Test Suite Optimization using Fuzzy Logic", 2012 International Conference on Emerging Technologies (ICET), IEEE December 2012

[9] https://www.tutorialspoint.com/junit/junit_tutorial.pdf

[10] Patroklos Papapetrou, "Code Coverage Tools (JaCoCo, Cobertura, Emma) Comparison in Sonar", December 19, 2012, https://onlysoftware.wordpress.com/2012/12/19/code-coverage-tools-jacoco-cobertura-emma-comparison-in-sonar/

[11] Hao Zhong, Lu Zhang, Hong Mei, "An Experimental Comparison of Four Test Suite Reduction Techniques", ICSE'06, May 20-28, 2006, Shanghai, China; 636-639.