ASSESSING SOFTWARE RELIABILITY USING GENETIC ALGORITHMS

R. Jain[*,a], and A. Sharma

Amity University Uttar Pradesh, Noida, Department of IT, Indira Gandhi Delhi Technical University for Women (IGDTUW), India.

ABSTRACT: The role of software reliability and quality improvement is becoming more important than any other issues related to software development. To date, we have various techniques that give a prediction of software reliability like neural networks, fuzzy logic, and other evolutionary algorithms. A genetic algorithm has been explored for predicting software reliability. One of the important aspects of software quality is called software reliability, thus, software engineering is of a great place in the software industry. To increase the software reliability, it is mandatory that we must design a model that predicts the fault and error in the software program at early stages, rectify them and then increase the functionality of the program within a minimum time and in a low cost. There exist numerous algorithms that predict software errors such as the Genetic Algorithm, which has a very high ability to predict software bugs, failure and errors rather than any other algorithm. The main purpose of this paper is to predict software errors with so precise, less time-consuming and cost-effective methodology. The outcome of this research paper is showing that the rates of applied methods and strategies are more than 96 percent in ideal conditions.

Keywords: Evolutionary algorithms; Genetic algorithm; Faulty /non-faulty data analysis; Software reliability engineering.

تقييم موثوقية البرمجيات باستخدام الخوارزميات الجينية

ر. جاين [أ،*] و أ. شارما [ب]

الملخص : الملخص: لقد أصبح دور موثوقية البرمجيات وتحسين جودتها أكثر أهمية من أي مسألة أخرى متعلقة بتطوير البرمجيات. فاليوم لدينا العديد من التقنيات التي تعطي تنبؤًا بموثوقية البرمجيات مثل الشبكات العصبية والمنطق الضبابي وغيره من الخوارزميات التطورية. و قد تم استكشاف خوارزمية جينية للتنبؤ بوثوقية البرمجيات. وكما نعلم , أن أحد الجوانب الهامة لجودة البرمجيات يُدعى موثوقية البرمجيات ، وبالتالي ، فإن هندسة البرمجيات تحتل مكانا مهما في صناعة البرمجيات . فمن أجل زيادة موثوقية البرمجيات ، يلزم أن نقوم بتصميم نموذجا يتنبأ بالخلل والخطأ في البرمجيات في المراحل المبكرة و يقوم بالعمل على تصحيحها وبالتالي بؤدي الى التحسين من أداء البرمجيات في أقل وقت وبتكلفة منخفضة. هذا ويوجد العديد من الخوارزميات التي تتنبأ بأخطاء البرمجيات مثل الخوارزمية الجينية ، والتي تتمتع بقدرة فائقة جدًا على توقع أخطاء البرمجيات والقصور والأخطاء أكثر من أي خوارزمية أخرى. و عليه فإن الغرض الرئيسي من هذه الورقة البحثية هو التنبؤ بأخطاء البرمجيات باستخدام منهجية عالية الدقة أقل استهلاكا للوقت , كما انها مناسبة من حيث التكلفة. هذا وتظهر نتائج هذه الورقة البحثية أن معدلات الوسائل والاستراتيجيات المطبقة لهذا الغرض هي أعلى من 96 في المائة من مثيلاتها في الظروف المثالية

الكلمات المفتاحية : خوارزميات تطورية؛ الخوارزمية الجينية؛ تحليل البيانات الخاطئة و غير الخاطئة؛ هندسة موثوقية البرمجيات.

Corresponding author's e-mail: jain.rac16@gmail.com

DOI: 10.24200/tjer.vol16iss1pp11-17

1. INTRODUCTION

Software quality prediction is a very vast and curtail issue in software improvement because it helps to find out various solutions for minimizing the cost of product development even if the product is time effective and needs fewer efforts to implement. There is a very much hot discussion on the notion that software quality characteristics (Briand. L. *et.al* 2000) cannot be directly measured. However, it can be measured with the help of other software attributes like coupling, size of the code, and of course, the complexity of the software product. There is a relationship between the attributes that are measurable or those which may be unmeasurable, though these software models are very inflexible to simplify and recycle on very innovative and undetected software as their accuracy failure has been found to be significant.

To overcome this process, this paper proposed a new method that, together with a genetic algorithm which acclimates such models, get to a new type of tests data. It gives an experimental evidence that clarifies this approach, which upbeats the decision trees machine learning algorithms (Quinlan, J.R 1993) like C4.5 and random presumption. To predict software reliability, we have to use software development models with respect to a software testing process. The parametric models are currently being used in software fault prediction, in which the most commonly used model is known as the Markov model. This model is sometimes also known as Non-Homogeneous Poisson Process (Wang J. *et.al* 2014).

Basically, Software Quality Assessment includes evaluating architecture results attributes and combination of these small attributes in order to achieve the best quality control and features. For this reason, the best way to go with software testing while using various test designs or models with Genetic Algorithm includes the white box testing methods and some other methods that are also available like Black Box testing method, robust testing, alpha and beta testing etc. In these proposed methods, a control structure of the software programme (that is actually a combination of the graph, use case diagram, and some other control flow diagram) has been prepared. Basically, a control flow diagram or control flow graph (Yamada S and Ohba 1983) is graphical representation where each node represents a program instruction and every edge represents a transfer of control between the instructions used for the same nodes using that particular edge. Many software reliability prediction and growth models have been designed and developed. We can use SPC (Statistical Process Control) for monitoring the process of Software Reliability and track all the actions which had been taken during the software failure process (Satya Prasad R. *et.al* 2012).

Now the role of genetic algorithm basically becomes more important because software testing is a key for software quality assessment. In this work, genetic algorithm and genetic programming are used to optimize the test data and generate reports on complex data analysis. For this reason, we use a fitness function that will be considered for following three criteria of population production as hereunder:

1. All path and traversal, that behaves as an independent path.
2. What percentage of an independent path is traversing?
3. The production time of the test data is not more than the controlled time.

2. BACKGROUND

Choi Sang-Hyeon and Lee Ikjin (2018), have given a detailed introduction about sequential optimization and reliability assessment. This is based on RBDO terminology, which has a higher success rate than conventional double loop RBDO methods although it is not more effective than the single loop approaches. Mishra, Dubey S.K. (2016) applied fuzzy approach for error detection and prediction. The authors used CK metrics for fault prediction and continuously made efforts to meet the challenge of maintaining software reliability in object-oriented languages with the help the fuzzy logic approach and CK metrics.

According to Amin A, Grunske L, Colman A. (2013) reliability is the main constraint software quality assessment. There were various models that have already been designed and used to estimate and predict the reliability based on the result of software testing methods. In this contrast the software reliability growth models are the most commonly used model to achieve this goal.

Sharma, Dubey S.K. (2015), conducted a survey on various techniques that extract from various journals and conferences and analyze the widely used methodologies and techniques used to predict software reliability. Kumar R and Gupta N (2015); in order to establish a relationship between reliability and complexity, as used the respective influence of the relationship between complexity and reliability.

Bishnu P.S and Bhattacherjee V (2011) proposed an idea about Kd-Tree methodology for predicting the fault at the early stage of software programming development with the help of the application of K-Medoids with KD-tree. They applied a learning approach for the prediction of faults in the software module. It is also called supervised technique that is also used for prediction effort.

Rauf *et. al*. (2010) have presented a GUI testing and analysis methodology on Genetic Algorithm. In order to expose the event-driven characteristics of GUI, they used event-flow graph technique together with automated GUI testing. The genetic algorithm uses various parameter combustions like coverage function, fitness function, crossover and mutation that ensure the correctness of test data.

For improving software testing efficiency, researchers Srivastava and Tai-hoon (2009) fond out the most critical paths after applying the genetic algorithm technique. To find the clusters in a program a method for optimizing software testing efficiency was identified. To achieve this, the researchers used WCF (weighted control flow graph) to test data generation by using the machine learning algorithm called genetic algorithm. There was another approach that found out the entire path in the underlying test called path testing that covers every possible path in the software program. The researchers concluded that, for improving software testing efficiency and performing exhaustive search, we can apply genetic algorithm easily and get meaningful results.

Yong C (2009) compared a test data for automatic path coverage by using genetic algorithm for generating new test data for path testing. He found that while using genetic algorithm less time is required for generating the meaningful test cased for path testing.

Ghiduk G (2008) introduced another method for reducing the software testing cost. It is a concept of relationships between the nodes of control flow graph that overcome the cost of testing the data. This is done with the help of a new fitness function that is defined by using dominance relationship for evaluation of the test data. The cost of testing is reduced by evaluating the effective fitness function and simultaneously comparing it to the random testing techniques for evaluating the effectiveness of the new fitness function as well as the methods that are used to overcome the cost of software testing. The proposed genetic algorithm and random technique were used with various object-oriented languages and programs, like structure oriented programming languages.

As proposed by Rajappa *et. al.* (2008), the use of graph theory with genetic algorithm approach will provide effective test cases for software testing. This is done by using the directed graph of all states of the program for the expected behavior, and the population was generated with the help of all nodes of the graphs. A combination of nodes was also selected from the population to perform crossover function and mutation for obtaining child nodes. This will lead to generating test cares in the real-time system.

Gupta, Rohil (2008) used the genetic algorithm methodology to generate test cases for an object oriented software program and other structured languages with the help of tree data structure. In an experimental step, this approach was used to generate test cases for JAVA classes. In the field of evolutionary algorithm set, the genetic algorithm is one of the most powerful and multi-purpose optimization tools which gives the right direction and recognition to the principles of the evolution. The genetic algorithm is very much capable of offering ideal solutions even in the most complex research environment De Jong *et al.* (1989).

At the beginning of 1960's, the genetic algorithm was first proposed by John Holland and at that time it was basically used for the issues that dealt with the complex search space programs and structural behavior Catal *et al.* (2009). The genetic algorithm was used to simulate the progression of living things through finding out a suitable answer related to all the problems related to it.

These issues necessitated the requirement of such type of algorithm that generates the populations of test, Goldberg D.E. *et. al.* (1992) analyzed a test or a series of tests, developing test data generators that can be structuralized as functional testing (Goldberg 1989; Dua *et. al.* 2002).

Basically, a genetic algorithm typically has five main parts as hereunder:

- Chromosome
- Pooling of Chromosomes
- Fitness Function
- Selection Function
- Mutation and Crossover Operator

We can understand chromosome as a string of binary data, or as a data structure. Initial Pool of chromosome can be manually created or may be randomly produced by some logic tricks and evaluation process. The selection function is responsible for deciding the initial progress stage of the genetic algorithm provided by the mutation and crossover operators in which chromosomes will participate. However, the crossover and mutation operators used to exchange the genes from two chromosomes and create two new chromosomes for generating population. A pseudo-code for genetic algorithm can be defined as:

```
GA      : Genetic Algorithm Starts
{
Initialize  : Initialize the population (Children)
Evaluate  : Calculate the fittest function applied
            on the population
If (Condition! = Accepted Criteria):  Implemen-tation
                of Loop until Termination Criteria
                reached
              {
               Selection Function:  Selection of
                                    Fitness function
               Operators          : Select operators
                                    like LRO (Linear
                                    range operators)
                                    Evaluation
               Process            : Applied fitness
                                    function on
                                    operators and
                                    evaluate results
               }
 }
```

Basically the genetic algorithm is a used for bounded and unbounded optimization.

3. GENERAL FAULT PREDICTION APPROACH

The defect test cases and test data regarding measurement that have been collected from software development efforts are used to construct a prediction model. In comparing the actual defectiveness and predicted defectiveness of the software modules in the test data, one can get the model performance. Figure 1 shows the common defect prediction procedure explained by Sunghun Kim *et.al*. (2011).

3.1 Tagging

The data defect can be collected for preparation of a prediction model. This is a process that used to extract the instances of data items from different s/w documentations.

3.2 Mining Features and Creating Training Sets

In this phase, there is one important thing that is taking place which is prediction of extraction of features and prediction of labels of illustrations both are coming from the prediction of fault test data. Defect prediction Sheta A (2006) has some complex components like metrics, keywords, dependencies and change that may be structural and nonstructural. The training set can be produced by combining the labels and features of instances that are used by the machine learning algorithms.

3.3 Constructing Prediction Models

We can use general machine learning algorithms Goldberg, D.E. (1987) such as support vector machines or Bayesian Network for building a training set and prediction model. This model can use two levels as "True" and "False" and obtain a new instance.

When the prediction of the model is calculated Pham *et al.* (1999) it requires testing data sets not training sets. This can be obtained by a prediction model that evaluates and compares the real labels and prediction. We can also separate the training sets from the testing sets by using the 10-fold cross-validation methods.

4. PROPOSED MODEL: A GENETIC ALGOROTHM APPROACH FOR FAULT PREDICTION

While using a genetic algorithm, we must use three components to create a prediction model that can predict the fault.

These are as follows:

1. Creation of a model of the software
2. Generating effective test cases
3. Calculation of software test adequacy

a. Generation of Test Data for test case generation

In this paper genetic algorithm is used to generate test cases that predict the software fault data set for prediction of software reliability Oliveira E *et. al.* (2005). For this reason, we assume that the chromosome is one that contains the input values. The proposed algorithm can evaluate the test data Berndt D *et. al* (2003) by implementing the program including the test data as inputs.

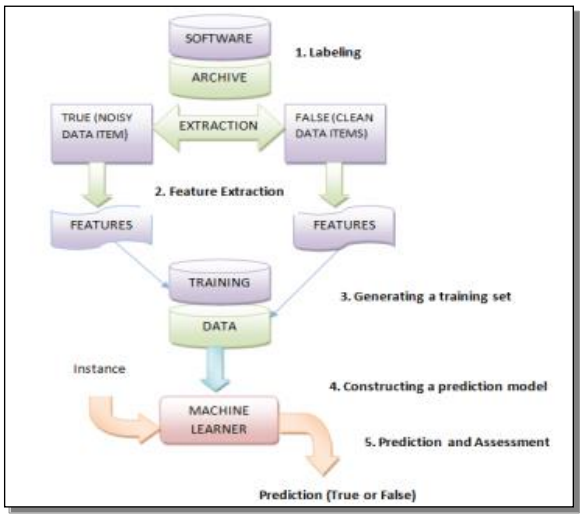

Figure 1. General fault prediction approach using machine language Sunghun Kim *et.al*. (2011).

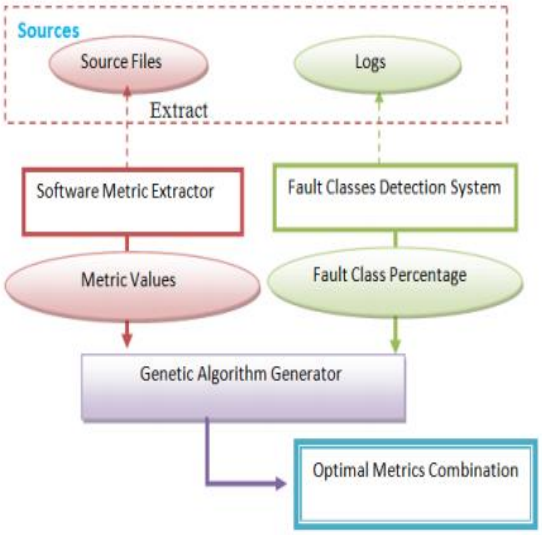

Figure 2. A genetic algorithm approch for software prediction.Variables Used:

Algorithm used: GenerateTData
Input: Part of a software program
CDG- Control Dependencies Graph for test Code
Ipol: Initial Population used for Genetic Algorithm
Test_R List of test requirements
Output:
Final: Set of test cases, used for fault prediction.
Variables Used:
CDG_Pth: A set of all paths in CDG
T_update: a record of satisfied test data

CDG_Pth: A set of all paths in CDG
T_update: a record of satisfied test data
Old_Pop & N_Pop: set of test cases
Motive: Which test cases are to be generated?
Max_A(): Function that returns Maximum attempts for single target
OFT (): Function that returns true when time limit reached and False otherwise.

Step: 1
Initialization and setup
Create software model
Create Control Dependencies Graph
Initialize initial population

Step: 2
Generate Test Cases
While ((some, (r, unmarked) Ɛ Test_R and not OFT ()) do
Select unmarked target from Test_R
While Target not marked and not Max_attempt () do
Compute fitness function
Sort the initial population according to the fitness
Select parents of new population
Generate new Populations
Execute program for each population
End while
End while

Step: 3
Generate the final effective test cases
Satisfy (Test_R)
Return (Final, Test_R)
End

The typical use of the Genetic Algorithm is crucial due to this procedure. This algorithm works according to the assigned goal and each test case has its own goal rather than having the fitness function unchanged during the test case generation Houck *et. al.* (1995).

The generation of test data has an input named as "Program" where a prototype can be used for generating the test data. Let us see the following code as an example code of a program to generate test cases using a genetic algorithm for software predictability and fault prediction.

Let us consider a programming code written in C-Language that has three integers i.e. x1, x2 and x3. Now generate test cases randomly and also create a CFG for this program code. The test cases are represented by T1, T2, T3, T4, T5, T6, T7 and T8.

1. Intiger (x1, x2, x3)
2. Read x1,x2,x3;
3. If(x1<x2)
4. If(x2<x3)
5. X1=x3;
6. Else x3=x1;
7. End if
8. End if
9. Print x1, x2, x3;
10. End

Suppose that the algorithm we are using selects the following test cases in order { (0, 6, 9), (0, 1, 4), (0, 1, 4), (0, 1, 4) }, then by using one point crossover method we can get T_5(1, 6, 4), T_6 (0, 1, 9), T_7 (0, 6, 4) & T_8 (5, 1, 4), after applying genetic algorithm, the two T_5 and T_7 satisfy the target and it is found that T_6 and T_8 are a faulty data set.

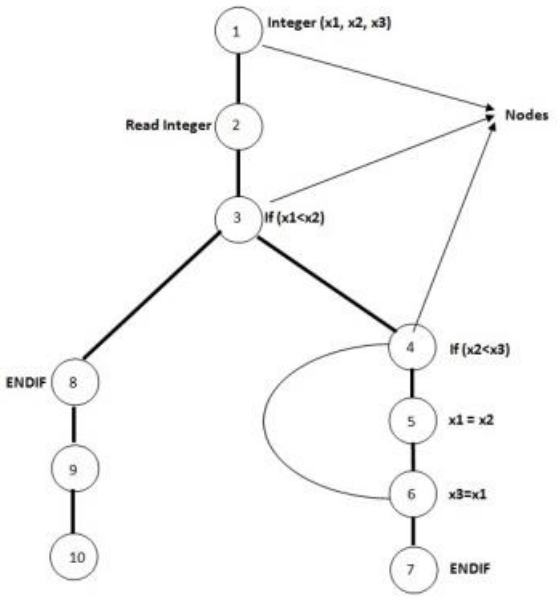

Figure 3. A control flow diagram for programming code written in C-Language.

After implementation of genetic algorithm we find following test data:

Table 1. Data set.

Test Cases	Input (x, y, z)	Statement executed
T1	0, 1, 4	1, 2, 3, 4, 6
T2	1, 6, 9	1, 2, 3, 4, 6
T3	5, 0, 1	1, 2, 3
T4	2, 2, 3	1, 2, 6
T5	1, 6, 4	1, 2, 3, 4, 6
T6	0, 6, 9	1, 2, 6
T7	0, 6, 4	1, 2, 3, 4, 6
T8	5, 1, 4	1, 2, 3

5. PROPOSED ALGORITHM

The proposed algorithm has been applied to the test data as per the follwing stepts written hereunder.

1. Take a program pseudo code written in any programming language (in this paper C Language)
2. Generate the test data of the program
3. Find all the possible conditions in the programing code.
4. Generate the random set of test cases and apply it,
5. Define the fitness function to calculate the path coverage of the programing code using CFG.
6. Check for the path coverage in CFG.
7. If path coverage is satisfied, then stop. Otherwise, go to next step.
8. Select the initial population with the rank based selection process.
9. Apply the GA's operations (crossover and mutation) to generate the new population.
10. Goto step 6

6. MEASUREMENT OF SOFTWARE RELIABILITY

Set of test-cases

The set of test cares are derived from data selection, which is a collection, of path traversal from one node to another.

Mutants test cases

To generate the mutants data, the volume of code along with mutant's operator are used.

The test adequacy can be calculated by the formula given below:

Adequacy = Coverage of test × the mutation score

The range of calculated adequacy is in the sort of [0, 1]; and it is useful in completing a accurate approximation of software reliability fault prediction. If the test model is an exact representation of the system under test, then the reliability of the software can be estimated by the following ratio:

$$\text{Adequacy} = \frac{\text{No. of testcases that is verified}}{\text{Total No. of testcases}} \qquad (1)$$

The program that is under the test can be assumed as a diagram shown (Fig. 2) containing the verified (specified by the symbol ⇒) and non-verified (specified by the symbol →) and arbitrarily persuaded fault is the dart thrown at it.

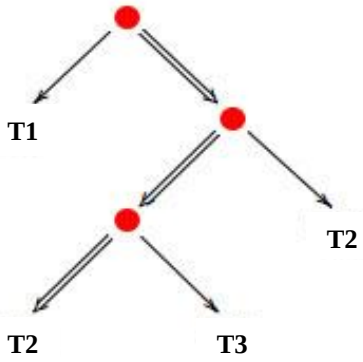

Figure 4. Example of path in a given program, where the path T2 is verified and reliable.

7. CONCLUSION

To achieve the software reliability, a code verification tool should comprehensively investigate every piece of code and verify its reliability against all possible data values. The study on programs has revealed the following results: Software reliability fault prediction is actually based on the number of defects that had already been present in the software before delivery or dispatch. While dealing with safety-critical software there was no situation found in which one can find the warnings or even errors during dynamic analysis. But in the present study, it was found that the average warnings or errors observed during the dynamic analysis decrease exponentially as the reliability increases.

CONFLICT OF INTEREST

The authors declare no conflicts of interest.

FUNDING

No funding was received for this research.

REFERENCES

Berndt D, Fisher J, Johnson L, Pinglikar J, Watkins A (2003), Breeding software test cases with genetic algorithms. *In System Sciences, Proceedings of the 36th Annual Hawaii International Conference on. IEEE* 338–347.

Bishnu P.S, Bhattacherjee V (2011), Application of K-medoids with kd-tree for software fault prediction. *IEEE Transactions Software Engineering* 321-452.

Briand L, Daly W, Wüst J, Porter D (2000), Exploring the relationships between design measures and software quality. *Journal of Systems and Software* 245-273.

CaoY, Hu C, Li L (2009), An approach to generate software test data for a specific path automatically with genetic algorithm. *International Conference on Reliability, Maintainability and Safety* 888-892.

Catal C, Sevim U, Diri B (2009), Clustering and metrics thresholds based software fault prediction of unlabelled program modules. *Sixth International Conference on Information Technology: New Generations* 199-204.

De J, Spears W.M (1989). Using genetic algorithms to solve NP-complete problems. *In ICGA* 124-132.

Dua A, Mishra G (2002), Stochastic search technique for solving constrained optimization problems with multiple objectives. *Proc of National Conference on Emerging Convergent Technologies and Systems* 399-404.

Goldberg D.E (1987), Genetic algorithms in search, optimisation, and machine learning. *Proc. of the 2nd Int. Conf. on Genetic Algorithms, Addison Wesley* 41 –49.

Goldberg D (1989), Genetic algorithms in search, optimization and machine learning. *Addison-Wesley, Reading, Massachusetts* 60-309.

Goldberg D.E, Deb K, Clark J.H (1992), Genetic algorithms, noise, and the sizing of populations. *Complex Systems* 333–362.

Ghiduk, Ahmed S, Girgis M.R (2010), Using genetic algorithms and dominance concepts for generating reduced test data, *Informatica Slovenia* 377-385.

Gupta N.K, Rohil M.K (2008), Using genetic algorithm for unit testing of object oriented software. *International Conference on Emerging Trends in Engineering and Technology* 308-313.

Houck Christopher R, Joines A, Kay J.G.M (1995), A genetic algorithm for function optimization. A *Matlab Implementation* 1-10.

Kim S, Zhang H, Wu R, Gong L (2011), Dealing with noise in defect prediction. *CSE"11, Waikiki, Honolulu, HI, USA* 481-490.

Kumar R, Gupta N (2015), Reliability measurement of object oriented design: Complexity Perspective. *International Advanced Research Journal in Science, Engineering and Technology* 38-44.

Mishra, Dubey S.K (2016), Reliability of object oriented software using fuzzy approach. *ISO/IEC 9126 model and CK Metrics* 398-401.

Nirpal, Premal B, Kale K.V (2010), Comparison of software test data for automatic path coverage using genetic algorithm. *Internal Journal of Computer Science and Engineering Technology* 42-48.

Oliveira E, Pozo A, Vergilio S.R (2006), Using boosting techniques to improve software reliability models based on genetic programming. *18th IEEE International Conference on Tools with Artificial Intelligence* 643-650.

Oliveira E, Silia C (2005), Modelling software reliability growth with genetic programming. *Proceedings of the 16th IEE International Symposium on Software Reliability Engineering* 39-121.

Pham H, Nordmann (1999), A general imperfect - software-debugging model with s-shaped fault-detection rate. *IEEE Trans. Reliability* 169–175.

Quinlan J.R (1993), Programs for machine learning. *Morgan Kaufmann Publishers* 235-240.

Rajappa V, Biradar A, Panda S (2008), Effective software test case generation using genetic algorithm based graph theory. *First International Conference on Emerging Trends in Engineering and Technology* 298-303.

Rauf A, Anwar S, Jaffer M.A, Shahid A.A (2010), Automated GUI test coverage analysis using GA. *7th International Conference on Information Technology New Generations* 1057-1062.

Sharma C, Dubey S.K. (2015), A perspective approach of software reliability models and techniques. *ARPN Journal of Engineering and Applied Sciences 7300-7308*

Srivastava P.R. Kim T.H (2009), Application of genetic algorithm in software testing. *International Journal of Software Engineering and its Applications* 87-96.

Sheta A (2006), Reliability growth modelling for software fault detection using particle swarm optimization. *IEEE Congress on Evolutionary Computation* 10428–10435.

Yamada S, Ohba (1983), S-shaped reliability growth modelling for software error detection. *IEEE Trans. Reliability* 475–484.