**International Journal of Engineering, Science and Mathematics**
**Vol. 6 Issue 3, July 2017,**
**ISSN: 2320-0294 Impact Factor: 6.765**
**Journal Homepage: http://www.ijesm.co.in, Email: ijesmj@gmail.com**

# GENERATION OF OPTIMIZED AND EFFECTIVE TEST CASE : A PROPOSED MODEL

Dr. Priyanka[1]
Dr. Rakesh Kumar[2]
Dr. Nipur[3]

**Keywords:**
Software Testing,
Test Case generation,
Optimized Test Cases,
Fault Toelerance

**Abstract**

This paper aims at developing a model for test case generation where optimized test cases are generated by making use of Genetic algorithm. These generated test cases are run on an implementation of the software under test and its mutants and output is generated. The goodness of result and effectiveness of test cases is verified by making use of an acceptor. Also in order to improve reliability of the system fault tolerance mechanism is implemented.

**Author correspondence:**

Dr. Priyanka
*Department of Computer Applications,*
*Maharaja Agrasen Institute of Management and Technology,*
*Old Saharanpur Road, Agrasen Chowk, Jagadhri, Haryana, India,*
gupta800@gmail.com

## 1. Introduction

Testing plays an important role in software development life cycle as it ensures the quality of software. But it's true that more than 50% efforts of development are spent in this activity [Tai(1980), Ince(1987),Graham (1992)]. And test case generation is the most important .Testing program against each possible input is infeasible because of large test case space. Exhaustive testing is not possible even for automatic testing technology. There is a need to hit test cases which provides maximum solution space. This can be achieved by optimizing the test case generation process. In literature several optimization techniques are available. Some of most prominent are Genetic algorithms, Particle Swarm Optimization, Ant Colony Optimization, Simulated Annealing, and Tabu [Elbeltagi et al.(2005)]. Any one of these methods can be used for optimizing the test case generation in the input space[Mc Minn (2004)].

These optimized test cases should be checked for their effectiveness.one method for testing the effectiveness of test cases has been suggested by authors in their previous work [Nipur et al.(2013 a)] . Mutation analysis can help ensure the worth of test cases and is defined as a fault based mechanism in literature[Demillo, Lipton, and Sayward (1978) And Budd, Lipton, Demillo, and Sayward, (1980)].

According to Morell (1990) Fault-based testing helps to find out the absence of pre specified faults in a program. Mutation-based testing works with a set of operators. Each of the operators modifies the source code as if an error has been injected. The modified program is known as a mutant. Test cases are generated and both mutants and original program are executed against them. If any test case can produce

[1]*Department of Computer Applications, Maharaja Agrasen Institute of Management and Technology, Old Saharanpur Road, Agrasen Chowk, Jagadhri, Haryana, India,*gupta800@gmail.com
[2]*Department of Computer Science and Applications, Kurukshetra University, Kurukshetra, Haryana, India*
[3]*Department of Computer Science and Applications, Kanya Gurukul Mahavidyalaya, Dehradoon, Uttaranchal, India*

different results then mutant is said to be killed. Otherwise, the mutant is live. Either the mutant program is found equivalent to the original program or the test data set is not adequate and needs enhancement. The adequacy of a test data set is measured by a mutation score (MS), which is the ratio of the number of killed mutants to the total number of non-equivalent mutants.

Mutation Testing is based on two basic Assumptions: (a) *The Competent Programmer Hypothesis*: In general programmers are competent. i.e., the programs they write are nearly correct. And the program which may be corrupted has only very small mistakes [Demilo(1978), Patrick(1997)]. (b) *The Coupling Effect Hypothesis*: Large program faults, which are semantic in nature depends upon smaller syntactic faults and can be detected with mutation testing. [Patrick(1997), How Tai Wah(2003)].

For Success of Mutation analysis as an effective tool for verification of effectiveness and completeness of test cases, a good quality killing of mutants is required. For that purpose this paper suggests an acceptor for the same purpose.

Also reliability is a must assured functionality of any working system. A system that fails on regular basis is not at all acceptable. High level of reliance in all application domains can be provided by Dependable systems. A dependable system is always Available (ready for use when needed), Reliable(able to provide continuity of service while it is used), Safe (does not have a catastrophic environment consequence on the environment), Secure(able to preserve confidentiality) as found in Gill (2002). Different approaches to achieve dependability (Fault Avoidance, Fault Removal, Fault Tolerance andFault Evasion) have been suggested in literature. Fault tolerance capability in a system manages to keep it operating, perhaps at a degraded level in the presence of faults [Gill (2002)]. Fault tolerance can be implemented by making use of N-Version and recovery blocks. Authors suggest impeding reliability in current work with the mechanism of fault tolerance in the form of recovery blocks.

Therefore, in this work a model for test case generation where optimized test cases are generated by making use of Genetic algorithm is suggested. These generated test cases are run on an implementation of the software under test and its mutants and output is generated. Thegoodness of resultand effectiveness of test cases is verified by making use of an acceptor. Also in order to improve reliability of the system fault tolerance mechanism is implemented.

## 2. Software testing as search based problem

There can be plenty of paths in a program when loops are there. Also paths become exponential when branches and conditions in them are considered. And also the number of test cases that may cover these paths are too much. Therefore test data generation problem can be considered a type of NP-hard or NP-complete problem. According to Harman &Jones(2001) any software engineering problem can be considered as a problem where competitive constraints needs balancing, inconsistencies need to be handled sometimes and many potential solutions are available, sometimes some good answers are recognizable but perfect are missing and moreover often precise rules for calculating the best solutions are not available. These attributes of software engineering problem makes it a candidate for meta heuristic search based solutions. Only the requirement is to represent the problem as a search problem.

For any software engineering problem to represent it as a search based problem following things are required: 1) representation where symbolic manipulations are possible 2) fitness number to calculate the goodness of a solution 3) operators that could be used to generate the next level of solutions [Harman &Jones (2001)]

In order to make use of meta heuristic search for finding solutions to software engineering problem its required that meta heuristic technique should perform better as compared to random search. Even if analytical techniques are available to find a solution search technique could be preferred as it may explore the whole solution space as compared to analytical. Solutions produced by analytical could be used as a seed for search based technique and better solutions could be explored [Harman &Jones(2001)]

### 3. Evolutionary and genetic algorithms

Evolutionary algorithms use simulated evolution to search for the solution of complex problems. They are used for search, optimization, machine learning and design problems. That's why they can be used even in test case generation in software testing field.

Genetic algorithms(GA) are most recognized type of evolutionary algorithms. GA was developed by John Holland and his students. GA is seen as a search process and also an optimization process [Whitley, 2001]. It can be used to solve non-linear, multi-modal and discontinuous optimization problems [Windisch et al. (2007)]. Selection of the fittest is the key for genetic search. In genetic algorithms generations are generated one after the other. Initially the base population becomes the first generation. From this generation fittest are selected using fitness proportional reproduction fi/f where fi is the evaluation of string I and f is the average evaluation of all strings. Strings having fi/f greater than 1 are treated as above average strings and are more likely to be selected and forms the intermediate generation which after implementation of recombination and mutation is converted to the next generation[Whitley(2001)]. At each generation selection is biased towards a better solution. Cycle is repeated till the acceptable level of optimal solution is achieved. Evaluation is problem oriented and relates directly to the structure of solutions [Pargas et al.(1999)]. Mutation helps in expanding the search space and some solutions may skip because of random nature of mutation. In initial phases it helps to avoid less likely solutions but during the final stages it weakens the stability of solution and hence affects the convergence speed [Li et al. (2010)].

Pseudo code for Genetic Algorithm[Goldberg(1989)]

```
Initialize (population)
Evaluate (population)
While (stopping condition not satisfied)
{
Selection (population)
Crossover (population)
Mutate (population)
Evaluate (population)
}
```

### 4. Use of genetic algorithm for generating test cases

Some work in use of genetic algorithms for generating test cases has been found in literature. Comparatively this is a newer area, still some researchers have contributed successfully in this domain. In case when GA has to be used in Test case Generation, the problem is to find a suitable and optimized test set that can satisfy some test adequacy criteria like path coverage, branch coverage, condition coverage, data flow coverage etc. for the software under test (SUT). The population in GA represents a Test Set for SUT. This set is selected randomly from the input domain of the SUT. GA has to evolve this test set to the optimized test set by means of the process described above in section 3, that will satisfy the criteria and hence become the solution to the problem. Some ideas are presented here:

In view of Korel (1990) Goal of any test generation problem is to generate an input x that belongs to the domain D and traverse the path P. this goal can easily be converted into sub goals and each sub goal can be handled using function minimization search technique. Branch predicate is a term related to each branch in a program that describes the conditions of execution of the branch. Each branch predicate can easily be represented by E1 op E2 where E1 and E2 represents arithmetic instructions and op represents any of {<, >, =,≠,≤, ≥} .This can be further converted into a predicate function expression $F(x_i)$ $rel_i$ 0 where $rel_i$ can be one of{< , ≤, =} and $x_i$ is any input from the input domain D. Now the value for each function F depends upon input variable x.

Now the problem of test data generation could be tackled like this. $X^0$ could be taken as a randomly generated initial input. The goal is to traverse the path P= <$n_{k1}$, $n_{k2}$, $n_{k3}$, ……$n_{kq}$ > where each $n_{ki}$

represents the nodes in control flow graph of the program and are linked to each other by edges between them. Now suppose $X^0$ has successfully executed the path P then $X^0$ is solution for the problem but if it could not completely traverse the path P but has traversed a part of it say $P_1 =< n_{k1}, n_{k2}, n_{k3}, \ldots\ldots\ldots\ldots n_{ki} >$ where $1 < i<q$. Now a sub problem originates. Branch function for above case fails at execution of $(n_{ki}, n_{ki+1})$ . Now purpose is to find the value of x such that F(x) becomes negative or zero and the path $(n_{ki}, n_{ki+1})$ is executed with the condition that $P_1$ is traversed on x. The process is repeated again and again till the goal of traversal of P is achieved. Now for finding the value of x is the basic search problem that has to be tackled. It has been solved by making use of Direct Search method which emphasizes on comparison of branch function values only to progress towards the minimum. Alternating variable method is deployed where value of a single variable amongst the input is varied and all others are kept constant. Further the help of exploratory search is taken in which values are incremented /decremented by very small values and direction for progress is found. When the direction is clear and base value of input variable for which the progress in function value is seen is available then pattern search is put on (large change in value) for accelerating the search process[Korel(1990)].

Roper et al.(1995) has experimented the idea on the branch coverage. They set aside the population randomly and then each test data (each individual) of the population is set as in input for the SUT. They noticed the coverage against each individual and assigned them fitness value depending upon the level of coverage one has done and implemented it in form probe value against each run. When all are run each have a fitness value associated. Then best fit individuals are selected either based on roulette wheel or tournament selection. And then crossover and mutation are implemented for next level of generation. When the combination of values in each temporary population gets the complete coverage of test criteria it become final solution otherwise the process is repeated.

According to Pargas et al.(1999) Initial population (test cases) may be generated randomly. Next level of population can be generated by making use of the fittest amongst the available. Fitness in the case of test generation can be handled by making use of CDG. An input is made to execute the program and the predicates that are actually executed in scenario of satisfying some testing criteria are noted and then they are compared with the set of predicates satisfying the same testing criteria for the same program. The input for which the matched predicates are maximum is fittest. A good collection of parents can be done on the similar pattern. Next generation is done by making use of cross over and mutation. Repeating the process can help in getting finally a good quality of test cases that help in satisfying a particular testing criteria. [Pargas et al. (1999)]

Sthamer (1995) makes use of domain testing by partitioning the domain into sub domains. Each sub domain corresponds to the part of the software and it contains inputs which are necessary to execute that path. Domain testing concentrates on domain errors which occur due to shifting of domain border due to wrong use of relational operators. Test data needs to verify the use of relational operators and the location of border segments. GA's will find out members of these sub domains. Better test adequacy criteria can be met by generating test cases near boundary values in order to confirm whether the boundaries are well placed. They suggested fitness function adapted from predicates which is a Boolean function of input variables, constants and relational operators. They aimed at identifying test cases which on change by small amount reverses the predicates from true to false and vice versa. These pivotal values become basis for fitness function. When choosing such type of test set with GA they will be capable of revealing more faults. Such type of data is called as boundary test data. The predicate function can be of form h(x) rop g(x) where x is the set of input values h(x) and g(x) may be complex functions and rop relation operators joining them. A possible fitness function can be $F=1/ (abs(h(x) - g(x))+a)^n$ where a is a numerical overflow adjustment and n can be 1,2,3. Generating data like this will ensure traversal of all branches. More close data is to boundary more exact information about the correctness of condition it will provide. Data near boundary will probably generate test data which crosses the boundary and covers opposite branch. Each branch will have different fitness function depending upon its predicate. Its sufficient to test branch single time because test data from same sub domain will follow same function and will follow same branch therefore all other tests from the same domain will not cover new faults.

Michael et al.(1997) presented an approach using GA based on the ideas of Korel's function minimization concept. Their approach was based on complete branch coverage unless the branch reach is infeasible. Michael et al.(1997) has suggested a method where a check is kept on reach of control on branch predicates with some input and further chaining that takes place and a table for the same can be maintained. If a branch has not been reached at all no function minimization will take place for that. Otherwise if all branches for the condition have been covered job is done. And if only one branch is covered function minimization for that predicate has to be done. Test cases has to be selected to cover that uncovered branch and further coverage effect for those test cases have to be marked in the table. This way heuristic use can help to reduce the efforts and hence enhances the efficiency.

Jones et al.(1996 ) used control flow graph instead of CDG . They used the branch coverage criteria for the purpose. For each test case they recorded the branch it has reached and covered. For each individual they generated the fitness value based on the branch as well as condition value of that branch.

Bueno(2002) worked on path coverage test data generation by making use of GA. They used predicates for path coverage and their execution for calculations of the fittest among the population. For better performance they used a past input data as an initial population.

Girgis(2005) has worked on coverage of def-use associations in the program for generating test cases by making use of genetic algorithm.

Rajkumari and Geetha (2011) have used memetic algorithm for optimized test case generation. They made use of schema theory of GA for the same. Their approach is a hybrid one. They used Global search to select global optima for the structural test and local search is used to select local optima for the sub branches.

## 5. Fault Tolerance Mechanism

Fault tolerance is a mechanism for hassle free working of a system even in existence of errors. System that fails usually is not at all acceptable in this competitive world. Fault tolerance provides the way of system recovery when failure occur and system continue in providing service [Sommerville(1999)]. Fault tolerance can be classified into single version and multiple version software techniques [Bharati (2003)]. Single version is one where fault tolerance capability is inherited in system through single implementation on the other hand multi version may have two or more versions for the same purpose. Most commonly used techniques for implementation of Fault Tolerance are N-Version Programming and Recovery Blocks [Sommerville(1999), Bharati (2003)]. Both techniques are based on design diversity where versions suppose to have different designs but provide same service.

### Recovery Blocks (RB)

This technique was suggested by Randell in 1970's [(Randell(1975)]. It is a design diverse technique where multiple versions of the software are executed in sequential fashion and is inherited by dynamic redundancy (stand by sparing) technique in hardware [(Chen, Avizienis (1978)]. On entry to system checkpoint is maintained to save the system state for backward recovery if required. It performs Fault tolerance after evaluating the results of outputs generated by primary alternate of software. On performing acceptance test on this output if results are well they are transmitted to environment otherwise if a failure occurs then system back recovers the state of the system which was maintained at the time of entry to recovery block and allows the execution of another available alternate. System continues to work like that unless either acceptance test has passed by some alternate or all the alternates have failed in such case failure message will be transmitted back to environment.

### N- Version Programming (NVP)

The NVP investigation project was started by Avizienis in 1975[(Chen, Avizienis (1978)] and is based on static (replication and voting redundancy) scheme from hardware. It's the approach in which more than two versions of software for the same specifications are done independently. They are supposed to be different

implementations with same functionalities. N versions of software units are combined to form N- Version software unit and a supervisory program to drive this unit is called as driver. This program is responsible for comparing the results of all versions and providing the status for the same. C vectors (comparison vectors) which are generated by the programs at cc points (cross check points) and the program state variables that need to be compared in c vectors at cc points are available at the specification of the software. Emphasis in NVP is to generate the N versions which don't have same errors. Therefore it's preferable they have different designs and even use different development process including language of implementation. NVP and RB both use the Execution Environment (EE) for implementation of Fault Tolerance. NVP uses a decision algorithm that is genetic in nature and is embedded part of EE not N- Version software unit. It compares the results of N versions and tries to find for consensus of two outputs and then the output is conveyed. In case of RB acceptance is part of RB system not EE and is particular for every application. All versions in case of NVP are executed parallel whereas sequentially in case of RB

## 6. Proposed Model For Test Case Generation

Here in this section author describes the proposed model for test case generation. For any problem initial set of test cases can be generated either randomly or if some test cases based on analytical approach method previously derived for the problem are available can be used. These test cases are optimized by making use of genetic algorithm. Many methods for TCG using GA have been suggested in the Literature as discussed in section 4. Any of these suitable methods can be employed. Author suggests to use function minimization method and its algorithm is presented here.

Algorithm test data generation with Genetic Algorithm using function minimization method

1 Set goal= traversal of path $<n_{k1}, n_{k2}, n_{k3}, \ldots\ldots\ldots\ldots n_{kq}>$

2 take an initial random input $X^0$ that belongs to the set of input Domain D put $X= X^0$

3 if X can traverse P then set solution= X otherwise X has traversed a sub path name it as $P_1 =< n_{k1}, n_{k2}, n_{k3}, \ldots\ldots\ldots\ldots n_{ki}>$ where $1 < i<q$. and find a new Value of X let $X_1$ that can make F(X)either negative or zero and hence can execute $(n_{ki}, n_{ki+1})$ with a constraint that $P_1$ is traversed by X. The process is repeated for a number of sub goals which either will be satisfied and hence will be able to find some X satisfying the P completely or otherwise some sub goal may not be solved which leads to no solution for the problem and no satisfying test case will be generated.

Application of this algorithm on initial set of test cases will now provide a set of optimized test cases (OTC).

For evaluating these test cases mutants of the software under test (SUT) are generated. These mutants can be generated by methods defined by authors in their previous work depending upon the type of application under consideration [Nipur et al.(2013 b), Kumar et al.(2010, 2009)]

SUT and its mutants are executed with the set of OTC generated as discussed above. Results of the generated outcomes from execution are recorded and verified by an Acceptor implemented for that specific purpose. Also killing of mutants is done based on the verification of this unit. If all mutants are killed then OTC are supposed to be effective and complete enough to find out the errors out of the SUT. Otherwise there is a need for enhancement of the set of OTC which directly means the method needs to be strengthened. OTC Set is enhanced and the process is repeated again and again unless until satisfactory results are achieved.

In order to enhance the reliability of the system fault tolerance mechanism is implemented by making use of Recovery blocks at the Acceptor unit. If principal unit of acceptor fails to produce the result of verification, another unit is called for its service. Like wise depending upon the availability of resources n number of recovery blocks could be set for the same.

Algorithm for proposed system

1) Test Cases (TC) are generated from input space (D) of SUT.

2) These TC are optimized by making use of Optimization Algorithm (GA) and OTC (optimized Test Cases) are generated.

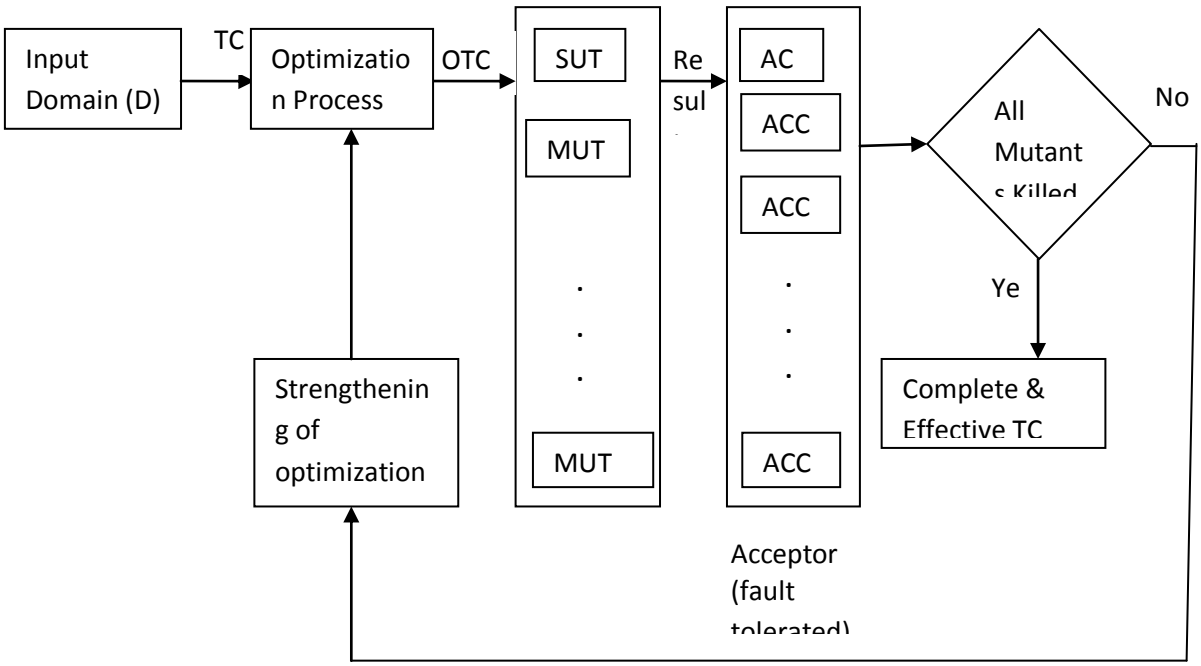
3) Mutants of SUT named MUT1, MUT2…….are generated.

4) SUT and Mutants are executed with OTC.

5) Results of execution are verified and Mutant killing is done by Acceptor (ACC).

6) In Case ACC fails System integrity is maintained by alternate versions of Acceptor (ACC1, ACC2……..ACCn).

7) If all Mutants are killed by Acceptor OTC set is complete and effective otherwise Strengthening of optimization process is required and steps from 2-6 are repeated.




**References**

Avizienis, A.(1985): N-Version Approach to fault tolerant Software. IEEE-Software Engineering, vol- SE11, No12, pp.1491 -1501.

Avizienis, A.(1995): The Methodology of N-Version Programming. Software fault tolerance, Edited by Lu, John Wiley and SonsLtd. pp. 23-46.

Bharathi, V. (2003): N-Version programming method of Software Fault Tolerance: A Critical Review . In conference proceedings, National Conference On Nonlinear Systems & Dynamics, NCNSD pp. 174-177.

Chen,L., Avizienis, A.(1978): N-Version Programming : A Fault-Tolerance Approach To Reliability Of Software Operation Reprinted From FTCSB 1978, pp. 3-9, Proceedings Of FTCS-25, Volume III.

Graham, D. R. (1992): Software testing tools: A new classification scheme, Journal of SoftwareTesting, Verification and Reliability, Vol. 1, No. 2, pp. 18-34.

Goldberg, D.E.(1989), Genetic Algorithms: in search, optimization and machine learning, Addison Wesley, M.A.

Girgis, M. R.(2005): Automatic Test Data Generation for Data Flow Testing Using a Genetic Algorithm.Journal of Universal Computer Science, Vol. 11, No. 6 pp. 898-915.

Harman,M. & Jones, B. F. (2001): Search Based Software Engineering. Information and software technology Vol 43(14) pp. 833-839.

Ince, D. C. (1987): The automatic generation of test data, The Computer Journal, Vol. 30, No. 1,pp. 63-69.

Jones, B., Stamer, H., Eyres, D.(1996): Automatic Structural Testing using Genetic Algorithms. Software Engineering Journal Vol 11 No 5, pp. 299-306.

Korel,B. (1990): Automated Software Test Data Generation. IEEE Transactions on Software Engineering Vol 16 No 8 pp. 870-879.

Kumar, R., Singh, S., Gupta, P. (2009): Mutation Testing in Database Systems. International Journal of Information Technology and Knowledge Management, January -June2009, Volume2, No.1 , pp. 217-222.

Kumar, R., Gupta, P. (2010): Mutation analysis and Database Systems. MAIMT Journal of IT and Management, May-October 2010, Volume 4, No.1, ISSN No.0974-066X, pp 82-96.

Michael, C.C., Mcgraw, G. E., Schatz, M.A., Walton, C. C. (1997): Genetic algorithms for dynamic test data generation.Technical report RSTR-003-97-11 version 1.1, RST Corporation.

McMinn, P. (2004): Search-Based Software Test Data Generation: A Survey, Software Testing Verification and Reliability.

Nipur, Kumar, R., Gupta, P. (2013 a): MutationTesting: A Tool To Evaluate Effectiveness Of Test Cases. MAIMT Journal of IT and Management, Nov-April 2013, Volume 6, No.2, ISSN No.0974-066X, pp 65-80

Nipur, Kumar, R., Gupta, P. (2013 b):Finite State Machine Based Modelling and Testing Of Web Based Applications using Mutation Analysis**.** Vivechan International Journal of Research, Vol4, pp.118-134

Pargas,R. P., Harrold, M. J.,Peck, R. R.(1999): Test- Data Generation using genetic algorithms. Journal of Software Testing. Verification and Reliability to appear.

Peng NIE (2012):A PSO Test Case Generation Algorithm with Enhanced Exploration Ability. Journal of Computational Information Systems Vol 8 Issue14 pp. 5785–5793.Available at http://www.Jofcis.com

Randell, B. (1975): System structure for Software Fault Tolerance. IEEE- Software Engineering, Vol. SE-1,pp.220-232.

Roper, M., Maclean, I., Brooks, A., Miller, J., Wood, M. (1995): Genetic Algorithm and the Automatic generation of test data Technical Report RR/95/195 [EFoCS-19-95], University of Strathclyde, Glasgow G1 1XH, U.K.

Randell, B., Xu, J.(1995): The Evolution of recovery block concept. University of Newcastle upon tyne.

Rajkumari, R., Geetha, B.G.(2011): Automated Test Data Generation and Optimization Scheme Using .International Conference on Software and Computer Applications IPCSIT vol.9, IACSIT Press, Singapore.

Sthamer, H. H (1995): The automatic generation of Software test data by using Genetic Algorithm. Ph.D Thesis,University of Glamorgan.

Tai K. C.(1980): Program testing complexity and test criteria, IEEE Transactions on Software Engineering, Vol. 6, No. 6, pp. 531-538.

Whitley,D. (2001): An Overview of evolutionary Algorithms: Practical Issues and Common Pitfalls. Information and software Technology special issue on Software on Software Engineering using meta heuristic Innovative Algorithms. Information and Software Technology, Vol.43 (14) pp. 817-831.