International Journal of Recent Research and Review, Vol. XII, Issue 1, March 2019
ISSN 2277 – 8322

Software Testing using Intelligent Water Drop (IWD) Algorithm

Baraa S. Alhafid, Laheeb M. Alzubedy
Software Engineering Department, Mosul University, Mosul, ZIP/+964, Iraq

***Abstract*—Software engineering plays a vital role in software testing process to avoid errors and defects in software before deliverer to the customer. To support all this, software rely on artificial intelligence techniques that are used in the development phases of software engineering, particularly in the software-testing phase, which gives the results of high quality and accuracy.**

***Keyword*—Software testing, intelligent water drop algorithm, swarms intelligent, basis path testing, white box testing**.

I. INTRODUCTION

Software test is the main approach to find errors and defects assuring the quality of software. Software testing is an expensive component of software development and maintenance. Testing is a complex, labor-intensive, and time consuming task thataccounts for approximately 50% of the cost of a software system development [4], incorporating the applicable criteria that follow components, Aim of the software testing is to uncover errors and faults present in the program, so that customer requirement can be properly fulfilled. Testing phase includes in the review of specification, analysis, design, and implementation part of the Software Development Life Cycle (SDLC)[7]. Due to the lack of cost and reliability, automation of testing process is necessary, so that the cost of testing can be reduced. Artificial Intelligence (AI) based techniques can help in removing this situation. AI based technique helps in solving the problem by using fast and proper judgments rather than using step by step deduction [2].In this paper, the tools and techniques of artificial intelligence were studied and employed in software engineering. And that is conducted through using the swarm algorithms (Intelligent Water Drop Algorithm) in generating independent path of the software written with C++ language in an automatic way because that enables the corporation which develops the program to save time and costs as well as ensuring the test process quality, which is estimated by 50% of the product cost. And in this paper, Independent Paths Generator Tool (IPGT) is constructed to generate independent paths in the programs it was also parser tool is constructed to generate control flow graphs for the programs that will be tested where control flow graphs helps In the perception of all the paths in the program unit which helps in the process of determine independent paths. The proposed (IPGT) tool succeeded in generating independent paths for several programs and in a very short time. The paper is structured as follows: section II introduces related work, section III present testing in software engineering with the objective and type of software testing, section IV describes the intelligent water drop algorithm, section V includes the Experimental Results, section VII includes Conclusions.

II. RELATED WORK

Various techniques have been proposed for automated testing to reduce efforts to a remarkable extent.

- Andreas W., Stefan W.,Joachim W.in (2007) suggested an empirical comparison of a genetic algorithm and a particle swarm algorithm applied to evolutionary structural testing. We selected 25 artificial test objects that cover a broad variety of search space characteristics (e.g. varying number of local optima), and 13 industrial test objects taken from various development project. The results indicate that particle swarm optimization is well-suited as a search engine for evolutionary structural testing and tends to outperform genetic algorithms in terms of code coverage achieved by the delivered test cases and the number of needed evaluations [5].
- Praveen R. S. , Tai-hoon K.in (2009) presents a method for optimizing software testing efficiency by identifying the most critical path clusters in a program. We do this by developing variable length Genetic Algorithms that optimize and select the

software path clusters which are weighted in accordance with the criticality of the path. Exhaustive software testing is rarely possible because it becomes intractable for even medium sized software. Typically only parts of a program can be tested, but these parts are not necessarily the most error prone. Therefore, they are developing a more selective approach to testing by focusing on those parts that are most critical so that these paths can be tested first. By identifying the most critical paths, the testing efficiency can be increased [12].

- Surender S. D. , Jitender K. C. , Shakti K. in (2010) presents an artificial bee colony based novel search technique for automatic generation of structural software tests. Test cases are symbolically generated by measuring fitness of individuals with the help of branch distance based objective function. Evaluation of the test generator was performed using ten real world programs. Some of these programs had large ranges for input variables. Results show that the new technique is a reasonable alternative for test data generation, but doesn't perform very well for large inputs and where constraints are having many equality constraints [11].
- Sanjay S., Dharminder K., H M Rai and Priti S. in (2011) presents a technique that based on a combination of genetic algorithm (GA) and particle swarm optimization (PSO), and is thus called GPSCA (Genetic-Particle Swarm Combined Algorithm) which is used to generate automatic test data for data flow coverage with using dominance concept between two nodes. The performance of the proposed approach is analyzed on a number of programs having different size and complexity. Finally, the performance of GPSCA is compared to both GA and PSO for generation of automatic test cases to demonstrate its superiority [4].
- Ahmed S. G., O.Said , Sultan AL j.in (2012) presentspaper deals with Automatic Generation of Feasible Independent Paths and Software Test Suite Optimization using Artificial Bee Colony (ABC) based novel search technique. In this approach, ABC combines both global search methods done by scout bees and local search method done by employed bees and onlooker bees. The parallel behavior of these three bees makes generation of feasible independent paths and software test suite optimization faster. Test Cases are generated using Test Path Sequence Comparison Method as the fitness value objective function. This paper also presents an approach for the automated generation of feasible independent test path based on the priority of all edge coverage criteria. Finally, this paper compares the efficiency of ABC based approach with various approaches [8].

III. TESTING IN SOFTWARE ENGINEERING

There are a many definitions of software testing, but one can shortly define that as: "A process of executing a program with the goal of finding errors". So, testing means that one inspects behavior of a program on a finite set of test cases (a set of inputs, execution preconditions, and expected outcomes developed for a particular objective [9]. There are three type of testing namely Black box Testing, White box Testing and Gray box Testing. In this paper a White box testing is used, White box testing based on an analysis of internal working and structure of a piece of software. White box testing is the process of giving the input to the system and checking how the system processes that input to generate the required output as illustrated in Fig 1 .It is necessary for a tester to have the full knowledge of the source code. White box testing is applicable at integration, unit and system levels of the software testing process. In white box testing one can be sure that all parts through the test objects are properly executed [10].

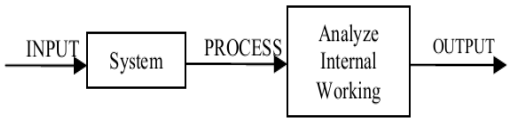

Fig. 1. Represent working process of White Box Testing

The types of white box testing techniques are [10] :

1. Control Flow Testing
2. Branch Testing
3. Basis Path Testing
4. Data Flow Testing
5. Loop Testing

- *The basis path testing*

Basis path testing is a white-box testing technique first proposed by Tom McCabe [13] and it allows the test case designer to produce a logical complexity measure of procedural design and use this measure as an approach for outlining a basic set of execution path (basic set is the set of all the execution of a procedure). These are test cases that exercise basic set will execute every statement at least once. Basic path testing makes sure that each independent path through the code is taken in a predetermined order. For this reason Basis Path Testing is used in this paper. The method devised by McCabe to carry out basis path testing has four Steps. These are [5]:

- Compute the program graph.
- Calculate the cyclomatic complexity.
- Select a basis set of paths.
- Generate test cases for each of these paths

A. Flow Graph Notation

Before we consider the basis path method, a simple notation for the representation of control flow called allow graph (or program graph) must be introduced, The flow graph depicts logical control flow using the notation illustrated in Fig 2, Each structured construct has a corresponding flow graph symbol [13].

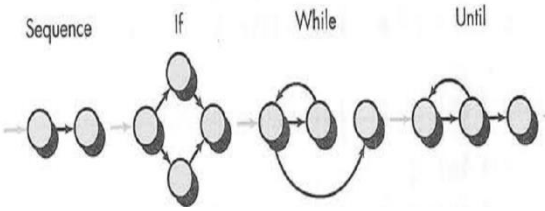

Fig. 2 Flow Graph Notation.

Control Flow Graph (CFG) describes the sequence in which the statements/instructions of a program are executed. It is representation of flow of control through the program. CFG is directed graph in which each node is a program statement/basic block and each edge represents the flow of control between statement/basic blocks. A basic block is a sequence of consecutive statements in which flow of control enters at the beginning and leaves at the end without halt or possibly of branching except at the end [8]. In a CFG, a node including condition is called a predicate node as shown in fig. 3, and edges from the predicate node must converge at a certain node. Area defined by edges and nodes is referred to as region [13].

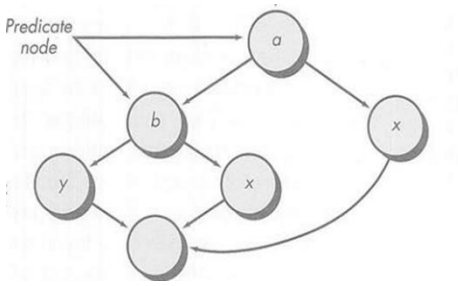

Fig. 3 Predicate node.

On a flow Graph as shown in Fig 4 :

- the symbol arrows called as Edges that represent the flow of control
- Circles are called as nodes, which represent one or more actions.
- Areas bounded by edges and nodes called regions

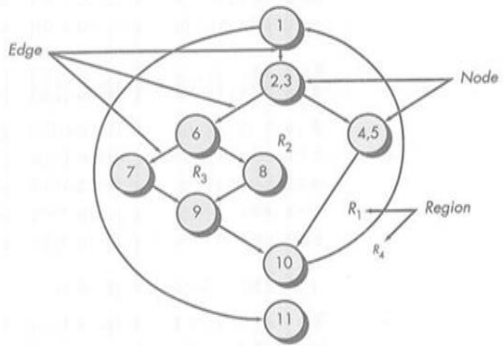

Fig. 4 Flow Graph .

B. Cyclomatic Complexity (CC)

The notion of Cyclomatic complexity was presented by McCabe. Cyclomatic complexity is software metric that delivers a quantitative degree of the logical difficulty of a program. Cyclomatic Complexity (CYC) is derived as the number of edges of the program's control-flow graph minus the number of its nodes plus two times the number of its linked components. Cyclomatic complexity purely depends on the Control Flow Graph (CFG) of the program to be tested [14] complexity is computed in one of three ways [13]:

- The number of regions of the flow graph corresponds to the Cyclomatic complexity.
- Cyclomatic complexity V(G) for a flow graph G is defined as V(G)=E-N+2
  Where E is the number of flow graph edges and N is the number of flow graph nodes.
- Cyclomatic complexity V(G) for a flow graph G is also defined as v(G)=P+1
  Where P is the number of predicate nodes contained in the flow graph G.

C. Determine Independent Paths

The value of V(G) Provides the upper bound on the number of linearly independent paths through the program Control structural .Through the Control flow graph in fig. 5 we expect to specify six Paths:

Path 1: 1-2-10-11-13
Path 2: l-2-10-12-13
Path 3: 1-2-3-10-11-13

D. Deriving Test Cases

Data should be chosen so that conditions at the predicate nodes are appropriately set as each path is tested. Each test case is executed and compared to expected results. Once all test cases have been completed, the tester can be sure that all statements in the program have been executed at least once [13].

IV. INTELLIGENT WATER DROP ALGORITHM

IWD algorithm [2-3] is a swarm-based optimization algorithm, simulated from observing natural water drops in river. IWD has been applied to various problems like Travelling Salesman Problem (TSP), N-queen puzzle ,Multidimensional Knapsack Problem (MKP) etc. These results have proved the significance of IWD algorithm over other swarm optimization algorithms. Another solution for TSP using IWD algorithm [2] is introduced where proposed algorithm converges very fast to the optimum solution.The improved IWD algorithm [4] has been applied to solve the air robot path planning in dynamic environments and results are quite impressive over genetic algorithm and ACO algorithm Since IWD has not yet been applied to the area of software testing and

Path 4: 1-2-3-4-5-8-9-2- . . .
Path 5: 1-2-3-4-5-6-S-9-2- . . .
Path 6: 1-2-3-4-5-6-7-8-9-2- . . .
The ellipsis (. . .) Following paths 4, 5 , and 6 indicates that any path through the remainder of the control structure is acceptable It is often worthwhile to Identify predicate nodes as an aid in the derivation of test cases. In this case, nodes 2,3,5,6 and 10 are predicate nodes [13].

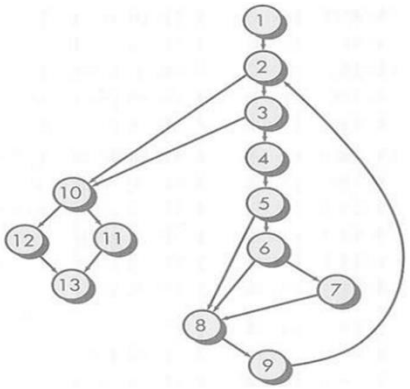

Fig. 5 Control Flow Graph

the effective results have been produced for various problems, this paper tries to derive a solution model for software testing using IWD in the hope that expected results will be more significant than the current solutions available for test data generation. Before moving to the proposed solution of IWD, general introduction is provided which describes its strategy along with available metrics in it.

IWD algorithm is a new swarm-based optimization algorithm inspired from natural rivers. In a natural river, water drops move towards center of the earth, due to some gravitational force acting on it. Due to this the water drop follows the straight and the shortest path to its destination [3]. Pictorial representation of basic IWD is shown in Fig 6. In ideal conditions it is observed that the optimal path will be obtained. Water drop flowing in the river has some velocity which is affected by another actor, i.e., soil.

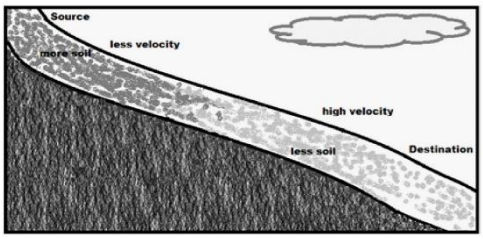

Fig. 6 Pictorial representation of IWD.

Some changes that occurred while transition of water drop from one point to another point are*:*

1. Velocity of water drop is increased*.*
2. Soil content in the water drop is also increased*.*
3. Amount of soil in the riverbed from source to destination get decreased*.*

Water drop in the river picks up some soil in it when its velocity gets high and it releases the soil content when its velocity is less [7] Some of the prominent properties of the natural water drop are taken, based on which IWD is suggested. IWD has the two following important properties:

1. The amount of soil the water drop carries, which is represented by Soil (IWD)
2. The velocity of water drop with which it is moving now, denoted by Velocity (IWD*)*

Value of both the properties may change during the transition. Environment contains lots of paths from source to destination [4] which may be known or unknown **.**When the destination is known, IWD follows the best path to reach the destination (best is in terms of cost and any other desired measure **.(**When destination is unknown it finds the optimal destination**.** From the current location to the next location Velocity (IWD) is increased by an amount **,** which is nonlinearly proportional to the inverse of the amount of soil between the two locations **,**referred to as the change in velocity. The Soil **(**IWD), is also increased by extracting some soil of the path between two locations. The amount of soil added to the IWD is inversely (and nonlinearly) proportional to the time needed for the IWD to pass from its current location to next location. IWD chooses the path with less soil content**.** In the proposed approach, IWD is applied over the Control Flow Graph (CFG) **.**The CFG depicts the logical control flow of the program [13]. All linearly independent paths could be obtained by CFG **.**Independent path is the path in the program that determines at least one new set of processing statement. In other words it introduces at least one new edge in the graph. Number of available paths can be obtained by finding the Cyclomatic complexity of the graph [13].

IWD algorithm has the following steps:

Step 1 : Initialize static and dynamics parameters
Step 2 : Put IWD on root node of the graph(CFG).
Step 3 : Calculate probability for choosing next node (j) from available pathof node (i).
Step 3.1: Probability can be find using formulae
In Eq.(1)

$$p(i,j)=\frac{paths(i)}{paths(j)}+\frac{soil(i,j)}{Init\_Soil}-\frac{\sum\frac{soil(i,k)}{Init\_Soil}}{No.of\ k} \quad \ldots. (1)$$

where, paths (i) = number of path from node (i) yet to be explored (CC(i)),∑soil(i,k) = Sum of the soil of every path i to k, i≠k.

Probability formulae is used for finding probability of a path when there are two nodes are avail-able for moving forward from the current node (i). This function can also tackle the blocked path situation. Along with that for handling the situation of paths having same CC, the concept of soil has also been introduced in the fitness function which is likely to be varied for the different paths .Step 3.2: Using the formulae as mentioned in Step-3.1, find probability for all outgoing paths from the current node (i)

Step 4: Choose next path which is having greatest probability because it is the optimal path where many other paths are yet to be explored, e.g., p(1,2) > p(1,3), then choose path(1,2) and add it to the visited path list.

Step 5: Update the Vel (IWD)(denoted by vel^{IWD}) moving from node (i) to node (j)as Eq.(2).

$$vel^{IWD}(t+1)=vel^{IWD}(t)+\frac{a_v}{b_v+(c_v\times soil^{2\times\alpha}(i,j))} \quad \ldots\ldots(2)$$

where, vel ^{IWD} (t+1) is the updated velocity of IWD.

Step 6: Update time parameter for IWD as Eq.(3)

$$time\left(i, j; vel^{IWD}(t+1)\right) = \frac{HUD(j)}{vel^{IWD}(t+1)} \quad \text{.....(3)}$$

HUD (j) = CC of CFG × CC at node (j) where, a local heuristic function HUD(j) has been defined for a given problem to measure the undesirability of an IWD to move from one location to the next.

Step 7: For the IWD, moving on path node (i) to node (j), it computes the change of soil as shown in Eq.(4).

$$\Delta soil(i,j) = \frac{a_s}{b_s + c_s.time^2\left(i, j; vel^{IWD}(t+1)\right)} \quad \text{……(4)}$$

Where, Δsoil(i,j) which IWD loads from path while traversing through that path Step 8. Update the Soil (IWD) (denoted by $soil^{IWD}$) by some amount which it has loaded from pathas shown in Eq(5).

$$soil^{IWD} = soil^{IWD} + \Delta soil(i,j) \quad \text{.....(5)}$$

Step 9: Update the soil of path between node (i) and node (j) as Eq(6).

$$soil(i,j) = soil(i,j) - \Delta\, soil(i,j) \quad \text{.....(6)}$$

Step 10: Repeat Step-3 to Step-9 until it encounters the end node or already visited node.

Step 11: Store the whole path as one of the final independent path in Path_List and decrease CC by 1

Step 12: Repeat Step-2 to Step-11 until, CC (root node) != 0 .

V. EXPERIMENT RESULTS

This section shows the result of the execution of algorithms, in order to determine the effectiveness and feasibility of the algorithm. Algorithms demonstrate that it is a reasonably accurate technique in terms of the paths covered using IWD Algorithm ,The (IPGT) applied in programs continues if-else states.

A. First program continues if-else state and the source code is shown below in Fig 7.

```
#include<iostream.h>
#include<conio.h>
void main()
{
/*this program about the condition if */
// enter the first no.
cin>>a;
// enter the second no.
cin>>b;
//enter the third no.
cin>>c;
 if(a>b)
    {
      if (a>c) {
                cout<<"the max no. is a="<<a;}
      else{
                cout<<"the max no. is c="<<c;}
    }
else
    {
      if(b>c) {
                cout<<"the max no. is b="<<b;}
      else
               { cout<<"the max no. is c="<<c;}
    }
}
```

Fig.7 source code of first program

This source code is chosen by IPGT and will generate independent paths for it and the result is :

1. Node number=12
2. Edge number=14.
3. Every edge in program and the nodes that make it and state that represent it as shown below in table I.
4. The Control Flow Graph (CFG) of the program that shown in Fig 8

TABLE I
Edges and Nodes of first program

States	Nodes	Edges
If	0 ---> 1	edge[1]
If	1 ---> 2	edge[2]
yes	2 ---> 3	edge[3]
else	2 ---> 4	edge[4]
Re.	4 ---> 5	edge[5]
Le.	3 ---> 5	edge[6]
else	1 ---> 6	edge[7]
If	6 ---> 7	edge[8]
yes	7 ---> 8	edge[9]
else	7 ---> 9	edge[10]
Re.	9 ---> 10	edge[11]
Le.	8 ---> 10	edge[12]
Re.	10 ---> 11	edge[13]
Le.	5 ---> 11	edge[14]

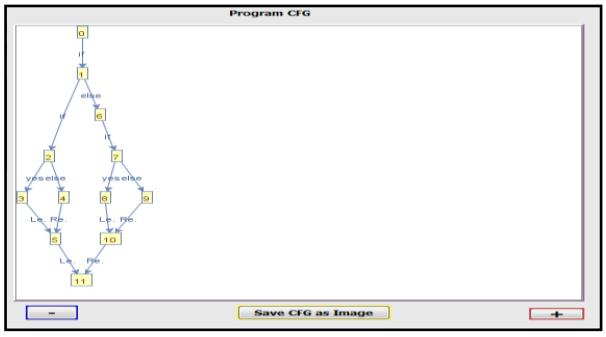

Fig.8 control flow graph of first program

5. The result when applied IWD algorithm in the first program is:
   - The no of independent paths is 4

   V(G)=14-12+2=4
   - The cyclomatic complexity of each node :

   node [0] = 4 , node [1] = 4, node [2] = 2, node [3] = 1, node [4] = 1, node [5] = 1, node [6] = 2, node [7] = 2, node [8] = 1, node [9] = 1, node [10] = 1, node [11] = 1.
   - The independent paths is :

   path [1] : 0 1 2 3 5 11
   path [2] : 0 1 2 4 511
   path[3] : 0 1 6 7 8 10 11
   path [4] : 0 16 7 910 11
   - Time is:0.0080 second.

B. Second program continues and the source code is shown in fig. 9.

```
#include<iostream>
void main()

{
    int Num;
/*enter the number  to check if it is even or odd*/
    cout<<"Enter Number to check it is Even or Odd: ";
    cin>>Num;
      if(Num%2==0)
        {
//if no. is even
            cout<<"Number is Even";

        }
      else
      {
//if no. is odd
          cout<<" Number is Odd";

      }
}
```

Fig.9 Source code of the second program

This source code is chosen by IPGT and will generate independent paths for it and the result is :

1. Node number=5
2. Edge number=5
3. Every edge in program and the nodes that make and state as shown in table II below.
4. The Control Flow Graph (CFG) of the program that shown in Fig 10.
5. The result when applied IWD algorithm in the second program is :

- The no of independent paths is 2

  V(G)=5-5+2=2
- The cyclomatic complexity of each node :

node [0] = 2, node [1] = 2, node [2] = 1
node [3] = 1, node [4] = 1
- The independent paths is :

path[1]: 0 1 2 4
path[2]: 0 1 3 4
- Time is:0.059 second

Table II
Edges and Nodes

Edges	Nodes	States
Edge[1]	0 --->1	if
Edge[2]	1--->2	yes
Edge[3]	1--->3	else
Edge[4]	2--->4	Le.
Edge[5]	3--->4	Re.

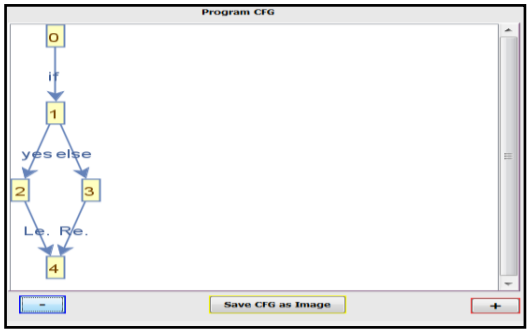

Fig. 10 control flow control of second program

VI. CONCLUSION

1. The automatically generation of test paths provides to software development team many advantages :
   - reduces the time taken to generate independent path in software testing.
   - reduces the effort and the burden for the team during the compilation of software testing .
   - reduces the cost for the software developer organization.
2. The use of artificial techniques to help software engineering gives a strong support to software engineering, particularly in the software testing phase.
3. Application of IWD algorithm in the generation of independent paths in the program have a great benefit in software Engineering.

VII. REFERENCES

[1] A. Windisch, S. Wappler, J. Wegener, "Applying Particle Swarm Optimization to Software Testing", ACM, 2007, pp.1121-1128.

[2] F. N. Raza, , "Artificial Intelligence Techniques in Software Engineering"", In Proceedings of the International Multi Conference of Engineers and Computer Scientists, Vol.2174,2009, pp.1086-1088.

[3] H. Afaq1 , S. Saini , " On the Solutions to the Travelling Salesman Problem using Nature Inspired Computing Techniques", IJCSI International Journal of Computer Science Issues, Vol. 8, No. 2,2011, pp. 326-334.

[4] S. Singla, D. Kumar, H M Rai, P. Singla," A Hybrid PSO Approach to Automate Test Data Generation for Data Flow Coverage with Dominance Concepts", International Journal of Advanced Science and Technology, Vol. 37, 2011,pp.15-26.

[5] H. Schligloff , M. Roggenbach, " Path Testing" , Advanced Topics in Computer Science: Testing, citeseer ,2002.

[6] H. Tahbildar ,B. Kalita , "automated software test data generation: direction of research", International Journal of Computer Science & Engineering Survey (IJCSES) Vol.2, No.1, 2011,pp. 99-120.

[7] R. S. Pressman , "Software Engineering A Practitioner's Approach, FIFTH EDITION",5th, McGraw-Hill Company,2001.

[8] Lam S. S. B., Raju M L H. P., Kiran M U., Ch S., Srivastav P. R.,2012, Automated Generation of Independent Paths and Test Suite Optimization Using Artificial Bee Colony, Elsevier , International Conference on Communication Technology and System Design,pp.191-200.

[9] Jovanovic , Irena," Software Testing Methods and Techniques",2008,pp.30-41.

[10] M. E. Khan, " Different Forms of Software Testing Techniques for Finding Errors", IJCSI International Journal of Computer Science Issues, Vol. 7, No 1, 2010, pp. 11-16.

[11] S. S. Dahiya, J. K. Chhabra, Sh. Kumar , " Application of Artificial Bee Colony Algorithm to Software Testin21st Australian Software Engineering Conference, ,2010,pp.149 IEEE -154.

[12] P. R. Srivastava , T. Kim," Application of Genetic Algorithm in Software Testing" ,International Journal of Software Engineering and Its Applications,Vol.3, No.4,2009,pp.87-96

[13] R. S. Pressman , "Software Engineering A Practitioner's Approach, Seventh Edition",7th, McGraw-Hill Company ,2010.

[14] S. Nidhra, J. Dondeti," BLACK BOX AND WHITE BOX TESTING TECHNIQUES–A LITERATURE REVIEW" ,International Journal of Embedded Systems and Applications (IJESA) Vol.2, No.2, 2012,pp.29-50.