AVISAR - A Three Tier Architectural Framework for the Testing of Object Oriented Programs.

Prashant Vats,
Research Scholar,
New Delhi, India.
prashantvats12345@gmail.com

Anjana Gossain,
USICT, GGSIPU,
New Delhi, India
anjana_Gosain@hotmail.com

Abstract: *In this research paper, we have proposed a three tier conceptual framework AVISAR for the testing of Object Oriented Programs. We have devised the proposed framework into three levels: Requirements Modeling, Test Case Generation and Effort Estimation. At the first level, we have used the Extend relationship of the Use Case to capture the events generated by the Classes during message passing between them through their objects. These events were captured by an Object Instantiator using Event Templates and further given to a Test case scenario generator, which generates the test cases. At the second level, the test cases will be generated based on the event templates that are used for capturing the events generated by the Extend relationship of the Use cases. Further, we have proposed a Genetic Algorithm (GA) for the effective test case prioritization to ensure maximum code coverage. At the third level, effort estimation, we have used the Cyclometric Complexity and Token count to perform the effort estimation for the Software under Test (SUT).*

Keywords: Object oriented, Genetic Algorithm, Fitness Function, Complexity.

I. INTRODUCTION:

The testing of a Software code is performed to detect the presence of errors in it. It's a universal fact that exhaustive testing of software is not possible, i.e. 100% code coverage cannot be achieved during the testing of an Object Oriented System (OOS) [1] because it requires that every statement in the given code and every possible path must be executed at least once.

During testing our main objective is to find situations that can produce errors and our aim is to detect the presence of errors in a given software code with the possible number of test cases that ensures that every Line of Code (LOC) has been executed at least once during the code coverage technique. We design the test cases with necessary preconditions, post conditions and required inputs and outputs to know about the results of those test cases. If there is any disperencies between the prescribed input and expected output, than the results are noted down, it shows the presence of error and is a call for error resolution mechanism. During testing the motive is to find the areas where the possibility of finding a fault is maximum. As because these areas are not easy to find so our motive should be on developing strategies and policies for choosing the effective software testing techniques. Also, very often terms like error, mistake, fault and bugs, failures are intermixed with various terminologies to define the process of testing. To overrule the error can be defined as any disperencies that is a causing a difference between the prescribed input and expected output. Further, if errors are present in a software code then it results into faults of the software. When this presence of errors also called faults gets executed then it is called as bugs, which may results into failures of the software.

On the contrary, while working with Object Oriented (OO) Methodology [2], it has becoming a challenging task to apply the conventional testing techniques to detect the presence of errors in an OOS. The OOS provides features like encapsulation, polymorphism that supports the software reuse. The OOS makes use of classes and their objects which provide classified and structured working mechanism. The objects of classes can communicate with each other through message passing and invocations. The conventional testing mechanism can't be directly applied to an OOS. In OO methodology features like encapsulation wraps a data into a single unit, which not only limits the extension on objects state but also restricts its intermediate test result analysis up to a certain level. Also, during inheritance in the OOS for testing a subclass, its inheritance structure needs to be broken down into a single class. When it is done, the testing efforts for the super class is not utilized so thus results into duplicate testing takes place. Further, In OOS, it becomes a quite tedious to obtain a snapshot of a class without creating extra methods that displays the class's state. Also, each new context of a subclass requires exhaustive regression testing because a method may be implemented differently using polymorphism.

In this paper we have presented a 3 tier conceptual framework AVISAR for the effective testing of OOS

978-1-4673-9080-4/16/$31.00 ©2016 IEEE

and for the generation of effective test cases that will ensure high accuracy with the maximum possible code coverage without any redundancy for Software under Test (SUT). With the help of using extend relationship of Use cases to generate test cases for an OOS, the test cases produced will form the basis of the test plans that would be fed to a test case scenario generator using event templates in the proposed conceptual framework which will operate on these test cases by using a Genetic algorithm (GA) in order to apply the mechanism of crossover, replication and mutation to produce a line up of effective test cases that will ensure the maximum code coverage for an given OOS.

II. GA INTO OOT

The main aim for the testing of an OOS is to automate the test case generation process for the OOS using an evolutionary strategy like Genetic Algorithm [3]. The use of GA in test case prioritization is helpful for the software testers for speeding up the process and in achieving the reduction in the testing efforts put by the Software developer. The use of GA based evolutionary testing approach helps in effective OO test-data generation and to addresses the problem of control flow accessibility in the flow path testing of the OOS. It also allows the test developer to achieve the test target and generating unit test cases in conceptual framework as compared with random testing approach. Using GA, it improvises the search for generating the test cases from event templates framed using the message passing between one object to the next, and provides us a better code coverage rather than as in the random testing. The main components of a GA are:

a. A chromosome which is a representation of a feasible solution to the problem and each of the chromosomal components is a gene;

b. A specific population of the encoded chromosomal component;

c. A fitness function for the chromosomal components which is used for the assigning of a scores for each of the each chromosomal component;

d. A selection of n number of operator to select parents according to their fitness;

e. A crossover operator for the chromosomes that generates new offspring by recombining the genetic sequence;

f. A mutation operator which differentiates the population by mutation in the chromosomes and presenting new off springs.

In this framework we have provided input as the extend relationships of the Use cases as a class path and a Java source file or a directory. A Junit test cases will be produced using an instance generator of classes that is based on the extend relation of the Use cases. It provides a generator for the means-of-instantiation by the objects to obtain a better code coverage for a given SUT. To address the issue of that how the unit tests cases for a SUT will be generated, For this problem we will use the GA and an event template which would be used to produce a Junit test cases by using Java which is very helpful for the expert software developers as well as for the novice software developers for designing the test cases for a SUT.

III. RELATED WORK FOR THE GA INTO OOT:

Authors Nirmal K. G., Mukesh K. R. [4] has proposed a method for generating test cases for classes in object oriented software using a GA programming approach involves Evolutionary Testing at the Unit level. In this approach the authors has used the tree representation of statements in test cases. It is implemented in Java.

Romain Delamare, Nicholas A. Kraft [5] has proposed an approach for the class integration test order problem in aspect-oriented OOS based on GA. It involves Fault based Testing. In this approach the authors have integrated the information on the relations between the classes & aspects of the SUT, as well as the information on the class methods is impacted by the aspects; is to produce an integration order that would avoid the modifications of the test case suites for the indirectly impacted classes for a SUT. But, this implementation is carried out on small scale projects.

Manoj Kumar, Mh.Husain [6] has proposed a GA for the evaluation of OOS by using class diagrams represented in the form of a tree. In this approach the authors has solved the problem of optimization & increases the efficiency of a system. This model also facilitates better memory management and code reusability.

Parvinder S. S., Satish K. D. [7] has proposed a GA based software fault prediction models with the Object-Oriented metrics. It involves Fault based Testing. The proposed system is adaptable to OOS and is useful in predicting fault prone classes for a SUT.

The Authors Jie F. and Lionel B., [8] has used inter-class coupling measurement & GA for minimizing the complexity of stubbing during integration testing. It is implemented with Java compilers.

Kobi Inkumsah, Tao Xie [9] has proposed a framework Evacon that integrates evolutionary testing & symbolic execution of objects for the Structural Testing of the OOS at the integrated class level. In this framework, the Search-based test Generation uses GA algorithms to find both the method arguments & method sequences for Software under Test (SUT). The tests cases generated by this

framework are helpful in achieving higher branch coverage as compared to the tests that have been generated by evolutionary testing, or symbolic execution of the test cases, or by performing the random testing within the same amount time.

M. Prasanna, K. R. Chandran [10] has proposed a model based approach for the automated generation of the test cases in an OOS using GA by analyzing the dynamic behavior of the objects due to internal and external stimuli. In this approach the authors uses the GA to generate test cases after the completion of the earlier conceptual design phase and the code errors could be detected at an early stage in the software development life cycle. It is implemented in Rational Rose.

Sangeeta Sabharwal, Ritu Sibal [11] has proposed a technique for prioritizing test case scenarios by identifying the critical path clusters using GA. They attempted to address the issue of requirements change by prioritizing the nodes of a control flow graph and State Dependency Graph using the stack based memory allocation approach and IF metrics. It is implemented in Rational Rose.

Gordon Fraser, Andrea Arcuri [12] has presented a search-based approach EVOSUITE, for optimizing whole test suites towards satisfying a coverage criterion, using GA. In this approach GA were used to handle the dependencies among predicates, & also in handling the test case length dynamically without applying exploration impending constraints.

A.V.K.Shanthi, G. Mohan Kumar [13] has proposed an approach using GA based data mining concepts for the generation of the test cases to provide all possible valid test cases of a given class diagram.

Margaritis B., Alexander C. [14] has use the GA for achieving a single-objective optimization for the attributes and methods placed in the classes of an OO system at Integrated Class Level testing. It involves Couple Based Testing. The said approach can be employed at the early stages of the analysis or design process when limited information about the methods or attribute dependencies is available but also at the later stages when a detailed class diagram or even the given source code for SUT is available to suggest possible improvements.

P. Maragathavalli, S. Kanmani [15] has proposed an approach called Class-Based Elitist based on GA to be used for testing the OOS. It involves the Evolutionary Testing of the OOS at the integrated class level for achieving better results over time. It not only speeds up the performance of the GA significantly and but can also be used for the real-time applications. The Elitism framework is implemented using Java.

Benoit B., Franck F. [16] has proposed their work on analyzing the application of GA algorithms for getting adapted to fit with the issue of test optimization. In their approach the authors has attempted to generates test cases instead of a set of test cases using the Mutation based testing to address the test case optimization, & memorizes efficient test cases from one generation to the next.

Joachim W., Kerstin B. [17] has proposed their work on evolutionary test environment with GA for the full automatic test data generation for most structural test methods for the testing of real-world SUT models. It enables the complete automation of test case design for various test objectives at the unit level testing. It also contributes to the quality improvement and to reduction of development costs for the SUT.

Zhongsheng Q. [18] has proposed their work for generating & optimizing test cases for a Web application testing based on user sessions using the GA at the Unit level Testing. It involves Specification Based Testing approach for capturing a series of user events, i.e., the sequences of URLs and name-value pairs in Web server (s). It then employs the GA in the reduction, grouping, prioritization and optimization of the test cases. This method is very effective when a large volume of users exist, and it is a Web application testing method of high efficiency.

IV. THE CONCEPTUAL SCHEMA OF THE 3 TIER FRAMEWORK AVISAR

In the conceptual 3 level testing framework AVISAR using the GA, its conceptual schema is categorized into three levels: Requirement Level, Test case Generation, Effort estimation. The Conceptual Schema of the 3 tier Framework for the testing of OOS is given in fig.1.

At the Requirement level, the requirements are feed to a Conceptual Model Generator which uses the UML modeling for the generation of the Use Cases with the extend relationships to further extend these USE cases. These extend relationship are then used as an object instantiator for generating the test cases for the SUT by using the event templates designed for these Use cases. The Classes in an OOS communicates with the others using the object instantiations and message passing with each other using objects. These objects in the extend relationships will frame the event templates to capture the events in the Use case diagram and will form the basis of generation of the test cases for the Classes in OOS. Further at the Requirement level these Use Cases can be further used to produce ER diagram, Data Flow Diagram Report for the Object Oriented Analysis (OOA) of the SUT.

At the Test Case Generation Level, Test case Scenarios are generated using the event templates generated by the USE cases. It provides the formation of test case reports to form the basis of the necessary

test plans and helpful in the preparation of the necessary test reports. Further, these test case sequence can be selected as a gene sequence to find its fitness function using the GA and can be used to perform genetic operation with the operators like cross over, mutation, and replication to find the effective test case sequence that will ensure the maximum code coverage.

At the Effort Estimation Level, the prioritized GA based test sequence will thus be fed to a Flow path testing approach for finding the Cyclometric complexity of the SUT. The Decision nodes and LOC is used to find the effort estimation for performing the testing of the SUT using Token Count and Estimated program length [19]. The use of the GA in the Path Testing at the Effort Estimations Level ensures the maximum code converge using the effective prioritized test cases based on the GA. These three levels in the AVISAR framework can be implemented using the Java language. For the generation of Use cases, UML can be used to provide Use cases with the Extend Relationship. These Extend relationships can then be fed to be used as an input to the Java input class which uses the J*instance_initiation* class to detect the event sequence generated by the "extend" to form the said event templates.

V. COMPONENTS OF THE AVISAR FRAMEWORK:

The framework is divided into three levels namely- Requirements Modeling level, Test Case Generation and Prioritization level, Effort Estimation level. The Components of the proposed framework is given in fig. 2.

At the first level, the user requirements are fed as input to the conceptual framework. Using the Requirements Modeling component, the requirement elicitation is performed. The requirements are classified into functional and nonfunctional requirements. By using the Conceptual Model Generator, the requirements fed are used for the conceptual modeling for the SUT using the UML. Further, it provides the Object Oriented Analysis (OOA) of the SUT. The Use Case generator provides the generation of the USE Cases with the Extend relationship for the SUT. The outputs provided at the first level are in the form of Use case reports and the conceptual modeling of the SUT provides the Class diagram, Sequential Diagram Reports. All these together will form the basis for further generation of ER diagrams, DFD's for the SUT.

At the Test case generation level, the Extend relationship in the Use case diagrams will form the basis for the generation of object instantiation between the Classes of the SUT. In OOS the Classes communicates with each other using the message passing objects. Thus then the extend relationship of the Use cases for the SUT will provide the event generation between message passing objects of the Classes of the SUT and will be used for the test case generation in the 3-Level conceptual framework using the Event capturing templates.. The output at this level is in the form of the set of test cases for the SUT.

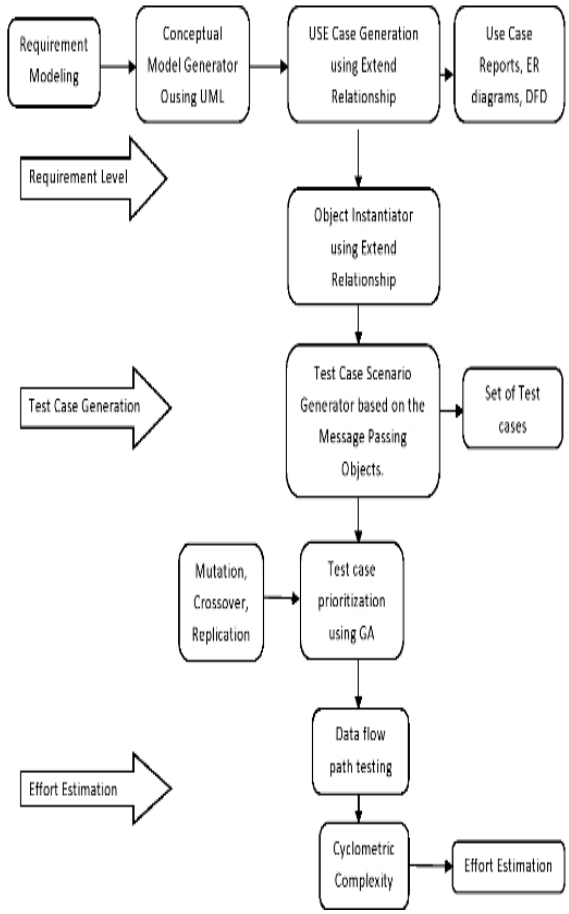

Fig.1. Data Flow Diagram for the Proposed 3 tier Framework AVISAR

For ensuring the maximum code coverage for the SUT, a sequence of suitable test cases will be chosen as a gene sequence for based on the selection criteria and a suitable GA operators like population, mutation, crossover, replication will be chosen to perform their intended function to find an effective gene sequence by determining its effective fitness function using the GA. It provides the efficient prioritization of the test cases. The minimal set of the effective prioritized test cases using the GA will eventually result in reducing the efforts for ensuring the maximum code coverage for the SUT. The GA for the conceptual framework works as follows:

Given A sequence i €n, i-> 1 to n,

For a given sequence i
from i=1 to n
Do
{
Randomly generate a population 'K' for a given set of 'n' individuals;
For (i=1; I<K;i++)
{
Evaluate the fitness function F (i) for
Each individual 'I' in the given set n;
}
Exit (if i=n); // stopping condition is fulfilled.
Repeat step 1 till I<- K.;

From the given set population K,
{
Select a pair of Parents P (J) using a recursive function Select_PairP ( );
For every pair of parent P (J)
Do
{
Perform combination & Mutation on P (J);
Population evaluation;
}
While
(Criteria of Termination is not satisfied)
}

Using the above mentioned GA, maximum code coverage can be ensured using the effective prioritized set of minimal test cases.
At the effort estimation level the prioritized test cases will be fed as input to the Flow Testing where all the possible code execution paths will be identified using the prioritized test cases to ensure that all the nodes are covered in the SUT. Further Cyclometric complexity [19] of the SUT will be calculated on the basis of the following rule:

$$M = E - N + 2P \quad \text{..... (1)}$$

Where E = the number of edges of the graph.
N = the number of nodes of the graph.
P = the number of connected components.

Further, Effort Estimation[19] can be done using the Token Count which is includes the size of the vocabulary of a program, which consists of the number of unique tokens used to build a program is defined as:

$$\eta = \eta 1 + \eta 2 \quad \text{..... (2)}$$

η: vocabulary of a program
$\eta 1$: number of unique operators
$\eta 2$: number of unique operands

The length of the program in the terms of the total number of tokens used is

$$N = N1 + N2 \quad \text{..... (3)}$$

N: program length
N1: total occurrences of operators
N2: total occurrences of operands

Volume of the Program can be calculated as follows:

$$V = N * \log 2\, \eta \quad \text{..... (4)}$$

The unit of measurement of volume is the common unit for size "bits"..
Program Level can be calculated as follows:

$$L = V^* / V \quad \text{..... (5)}$$

Where value of L.
i.e. --> 0< =L<=1;
if L=1 then the OOP is written at the highest possible level
LOC for OOP ← (i.e., with minimum size).
Program Difficulty can be calculated as follows:

$$D = 1 / L \quad \text{..... (6)}$$

As the volume of an implementation of a program increases, the program levels decreases and the difficulty increases.
Effort can be calculated as follows:

$$E = V / L = D * V \quad \text{..... (7)}$$

The unit of measurement of E is elementary mental discriminations.
Estimated Program Length can be calculated as follows:

$$N = \eta 1 \log \eta 1 + \eta 2 \log \eta 2 \quad \text{..... (8)}$$

VI. EXPERIMENTAL RESULTS

The use of GA for the testing of OOP in the framework AVISAR has provided improvised results as compared to the other techniques in terms of test case prioritization. The results are shown in Fig. 3.

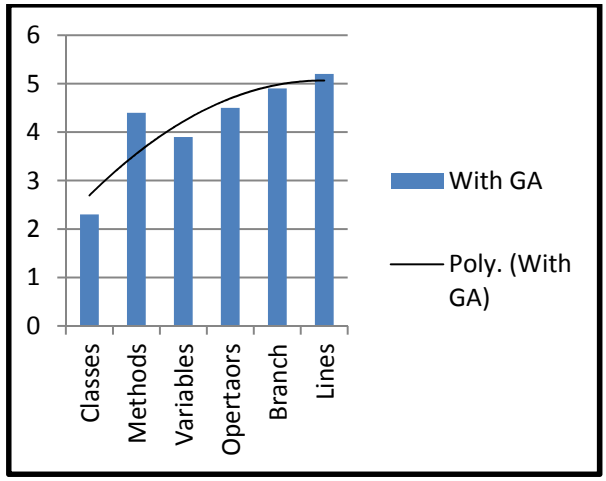

Fig. 3. Improvised Test case prioritization using GA.

Further also, when a comparison is made for the effective code coverage for the SUT using GA and without using GA, we have obtained improvised results using GA as compared to the code coverage without using GA.

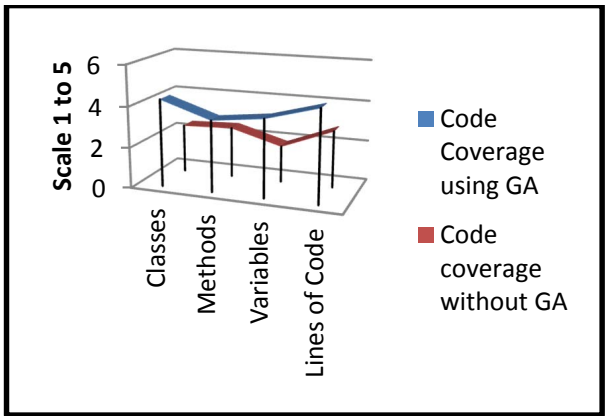

Fig. 4. Comparison using GA for the Code Coverage.

VII. CONCLUSION & FUTURE WORK.

In this research paper we have presented a 3-tier conceptual framework AVISAR using the GA for the testing of OOP. After carrying out our experimental results we have observed that the use of GA in the test case prioritization has provided us improvised results as compared to other approaches. Also, it has provided us effective code coverage during. Further also, we have given the effort estimation for the said SUT to reduce the efforts of the developer. As future work, the proposed conceptual framework can be implemented as an automated tool to find its application into the Software industry.

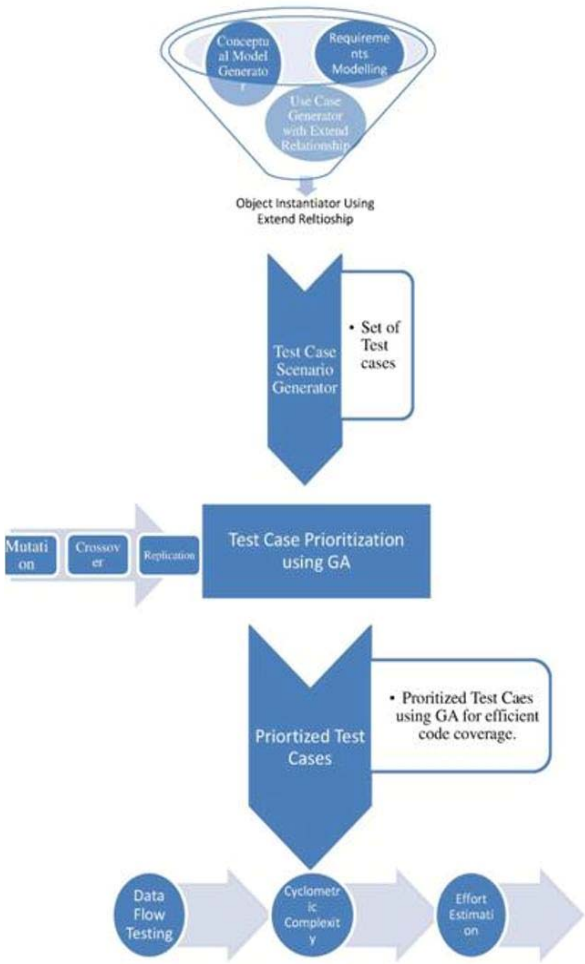

Fig. 2. Components of the 3 tier Architectural Framework AVISAR.

REFERENCES:

[1] Smith, M.D. (1992), "A Framework for Testing Object-Oriented Programs", *Journal of Object-Oriented Programming*, June 1992, pp. 45–53.

[2] Chan, W.K., Chen, T.Y. and Tse, T.H. (2002), "An Overview of Integration Testing Techniques for Object-Oriented Programs", *Proceedings of the 2nd ACIS Annual International Conference on Computer and Information Science (ICIS 2002), International Association for Computer and Information Science*, Mt. Pleasant, Michigan (2002).

[3] Godberg, David E. (2009), *Genetic Algorithms in Search, Optimization and Machine Learning*, Published by Pearson Education, 4th Edition.

[4] Nirmal, K.G. and Mukesh, K.R. (2009), "Using Genetic Algorithm for Unit Testing of Object Oriented Software", *Pub. in IJSSST*, Vol. 10, No. 3, pp. 99–104.

[5] Delamare, Romain, Kraft, Nicholas A. (2012), "A Genetic Algorithm for Computing Class Integration Test Orders for Aspect-Oriented Systems", *Pub. in IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST)*, pp. 804–813.

[6] Kumar, Manoj and Husain, Mh. (2011), "An Efficient Algorithm for Evaluation of Object-Oriented Models", *Pub. in International Journal of Computer Applications,* Vol. 24, No. 8, pp. 11–15, June 2011.

[7] Sandhu, Parvinder S. and Dhiman, Satish Kumar (2009), "A Genetic Algorithm Based Classification Approach for Finding Fault Prone Classes", *Pub. in Proceedings of World Academy of Science, Engineering and Technology*, Vol. 60, pp. 485–488.

[8] Briand, Lionel C. and Feng, Jie (2002), "Experimenting with Genetic Algorithms and Coupling Measures to Devise Optimal Integration Test Orders", Pub. in Carleton University, Technical Report SCE-02-03, Version 3, October 2002.

[9] Inkumsah, Kobi and Xie, Tao (2008), "Improving Structural Testing of Object-Oriented Programs via Integrating Evolutionary Testing and Symbolic Execution", *Pub. in Proceedings of 23rd IEEE! ACM International Conference on Automated Software Engineering*, pp. 297–306.

[10] Prasanna, M. and Chandran, K.R. (2009), "Automatic Test Case Generation for UML Object Diagrams using Genetic Algorithm", *Pub. in Int. J. Advance. Soft Comput. Appl*., Vol. I, No. I, pp. 19–32, July 2009.

[11] Sabharwal, Sangeeta and Sibal, Ritu (2011), "Applying Genetic Algorithm for Prioritization of Test Case Scenarios Derived from UML Diagrams", *Pub. in IJCSI International Journal of Computer Science Issues*, Vol. 8, Issue 3, No. 2, pp. 433–444, May 2011.

[12] Fraser, Gordon and Arcuri, Andrea (2011), "Evolutionary Generation of Whole Test Suites", *Pub. in Proceedings of 11th IEEE International Conference on Quality Software (QSIC)*, pp. 31–40.

[13] Shanthai, A.V.K., Kumar, G. Mohan (2011), "Automated Test Cases Generation for Object Oriented Software", *Pub. in Indian Journal of Computer Science and Engineering*, Vol. 2, No. 4, pp. 543–546.

[14] Margaritis, B. and Alexander, C. (2010), "Placement of Entities in Object-oriented Systems by Means of a Single-objective Genetic Algorithm", *Pub. in Proceedings of Fifth IEEE International Conference on Software Engineering Advances* (ICSEA), pp. 70–75.

[15] Maragathavalli, P. and Kanmani, S. (2012), "Multi-objective Genetic Algorithm using Class-Based Elitist Approach", *Pub. in Computer Science & Engineering: An International Journal (CSEIJ)*, Vol. 2, No. 5, pp. 3I–41 , October 2012.

[16] Baudry, Benoit and Fleurey, Franck (2005), "From Genetic to Bacteriological Algorithms for Mutation-based Testing", *Pub. in Journal of Software Testing, Verification and Reliability*, Vol. 15, pp. 73–96.

[17] Wegener, Joachim and Buhr, Kerstin (2002), "Automatic Test Data Generation for Structural Testing of Embedded Software Systems by Evolutionary Testing", *pub. in proceedings of ACM GECCO '02 Proceedings of the Genetic and Evolutionary Computation*, pp. 1233–1240.

[18] Zhongsheng, Q. (2010), "User Session-Based Test Case Generation and Optimization Using Genetic Algorithm", *Pub. in J. Software Engineering & Applications*, Vol. 3, pp. 541–547.

[19] Aggarwal, K.K., *Software Engineering*.