Estimation of Efforts During Testing of OOP using The AVISAR Framework

Prashant Vats, Manju Mandot

Abstract*: *In this research paper we have proposed our research work on the evaluation of efforts during the testing of OOP using the AVISAR object oriented testing framework which is based on the Genetic Algorithm. The proposed framework AVISAR provides a platform to address the issues which are related to the testing of Object Oriented aspects of OOP like Polymorphism, Inheritance, Polymorphism etc. which plays a key role in Effort Estimation of an Object Oriented Software.

***Keywords*: *Genetic Algorithm (GA), Inheritance, Polymorphism, Object Oriented Programs (OOP), Object Oriented Software under Test (OOSUT*)**

I. INTRODUCTION

An object-oriented testing framework should be able to distinguish between dissimilar entities. During testing the test coverage criteria plays a vital rule for selecting the test and at what time to stop it. When estimating OOS effort, the control flow chart approach also does not seem to be helpful as an OOS abstraction. Throughout the production and testing of OOS, the central problem with the application of conventional metrics is that it cannot determine the SUT complexity using the control framework. In most OOS, the methods called in classes that use objects in a system under test are so small that it is almost impossible to resolve the number of decisions in the control flow chart when these OOPs are tested for a SUT. The use of the GA-based rolling test approach tackles the control flow accessibility problem in Object Oriented Software's flow path tests and helps the effective generation of Object Oriented test data. It also aids the test developer in achieving the methods call patterns between Classes during the object call by procedures between these Classes to evaluate these OOPs to find out that which abstractions in these classes are important to evaluate object-oriented the software. Generally, the use of any OO metric means a broad technique to assess the estimate of effort while testing classes in an object-oriented program. But in addition, there are more complexities involved in these metrics during object call patterns between call procedures to access the different classes of a system under test. So in this paper we have proposed A methodology based on the use Genetic Algorithm for the effort estimation during the Testing of OOP using the AVISAR frame work [17].

Revised Manuscript Received on April 15, 2020.
* Correspondence Author

Prashant Vats*, AIMACT, Banasthali Vidyapith, Rajasthan, India. Email: prashantvats12345@gmail.com
Manju Mandot, Professor, J. R. N. Rajasthan Vidyapith, Udaipur, Rajasthan, India. Email: Manju.mandot@gmail.com

II. NEED FOR EFFORT ESTIMATION IN OOP

To ensure the quality of a product, a better understanding is provided by the "object oriented metric" (OOM) for Object-oriented Software under Testing (OOSUT). It offers the needed access to process efficiency to improve the quality of work done at the project level. Object Oriented Features like such as information hiding, encapsulation, inheritance, and the object abstraction approach lead to the requirement for a specialized metric applied to OO systems to estimate effort during the testing. A well-designed OO metric must be able to meet the requirement for an indication of the extent to which concealment was attained in an "OO design" (OOD) to make sure OOD quality during the OOP test.

III. RELATED WORK IN OOT

Hiroki T., et al. [1] utilized formal tests based on a Unit-class Petri color network to analyze the present behavior of objects. This approach is does not depend upon specific requirements, design methods, and languages. Jeff O., et al. [2] used peer-based tests at the level of class integration to resolve the problem CITO ("Class Level Integration Test") with edge weights taken from quantitative coupling. By automating the CITO problem, the capacity of developers has improved, but at the same time it cannot be applied to bigger systems is one of the limitations. R.B. Borie, et al. [3] used class-level Flow Graph-based tests to allow automatic test cases generation from properly stated classes. In it, they initiated the building of a simple flowchart that also aids generation of test cases and coverage analysis. However, alternatively, it may not adequately examine class behavior as it does not model the space of class state. Taratip S., et al. [4] used class-based cut-off tests to find test sequences to assuage the use of test pieces. Its key benefit is that it causes more failures to be detected in a much shorter time; however it cannot be scaled for big systems. Frankl, P., et al. [5]. employed the class-level ASTOOT technique (i.e. "a set of tools for OOT") to produce test cases that observe class state transitions and state values. In this technique, the test execution system can automatically verify the accuracy of the test cases with a test oracle. However, because test cases are randomly produced, analysis of test coverage is hard to carry out and the anticipated outcome of test cases can be specified on a "Boolean" value label. Dasiewicz, P., et al. [6] employed system-level event flow tests to prepare test cases that highlight the interaction between interrelated events.

The detection of latent loops and restricts the loop path by manual intervention is its main advantage. Since it makes use of telephone "private branch exchange" (PBX) software, it is not appropriate for systems of small-scale. Parnas, D.L., et al. [7] employed class-based graph-based tests to replicate class behavior like a directed graph known as "the test graph". A test oracle can also be an application of this test graph. Manuel O., et al. [8] proposed a randomized Adaptive object-level test to assess the effectiveness of a test approach called "ARTOO" to discover actual flaws in actual software's. Wang, Y., et al. [9] to effectively state the behavior of classes or software modules, proposed "Trace specification rewrite" approaches for class-level testing. Provides canonical traits assigned directly between the crawl space and the class operation space. T.Y. Chen, et al. [10] has declared the state-based tests at the integrated class level to signify the changing states of the SUT. Use FSM to model an integrated system. Other test approaches proposed by the same authors [11] are tests based on peer-based events, which use the relationships between events to take advantage of the systems and verify the violation of restrictions. Tom Maibuam, et al. [12] employed tests based on the Integrated Class Level Coordination Contract to present a technique to implementing test cases with the Coordination Agreement concept. Through the use of contract, generation and execution of test cases can be automated. A. Jefferson, et al. [13] used integrated class-level category splitting techniques to demonstrate that present approach can discover flaws in OOS; the combination of the "category partition method" and a tool to detect memory management failures are extremely useful for OOT. They inspected a technique based on specifications with two small programs. Notkin, D., et al. [14] employed tests based on unit-level statistical algebraic abstractions, without requiring any specification, to automatically identify common and special unit tests for a class. It presents the portrayal of program behaviors and the detection of common and special tests through the use of statistical algebraic abstractions. M. Burrows, et al. [15] proposed another approach known as "Eraser" to dynamically detect data executions in multi-threaded block-based programs. It imposes a simple blocking discipline rather than seeking careers in general parallel programs. André B., et al. [16] proposed an algebra-based approach to the CSP process for class-level testing. Java is used for its implementation and it handles single processor and multiprocessor environments and also meets real-time priority programming requirements. For the object oriented paradigm, this technique simplifies the use of priorities.

IV. EFFORT ESTIMATION USING AVISAR

The proposed OO metric tool in the OOT framework "AVISAR" consists mainly of three components: (i) Complexity calculator, (ii) Cohesion calculator, (iii) Estimator for OOD (OODE). With the use of these 3 components, it works collaboratively. The complexity degree involved during the various methods will be calculated by the Complexity calculator component, which will involve the class and individual weights assigning for these classes. The OODE will provide the measurement of the degree of characteristics such as encapsulation polymorphism, & inheritance in an OO class. In addition, the cohesion component calculator provides the measurement of the extent of cohesion in a SUT. Using its all elements together to estimate the overall effort during testing and the development of an OOS. In this framework, for a given OO code block, the Lines of Code (LOC) number is fed to the input of a Requirement Model that lay out the NOC (number of classes) to forming the base for the estimation of the efforts in the proposed OOS. In addition, these NOCs are saved in requirements for the data dictionary storage that will provide as container for storing the requirement details for these NOCs produced for the provided code.

4.1 Cohesion Extent Calculator: As input, it accepts LOC. Using these LOC, a bipartite graph is formed. The functions and attribute called from a class in an OOS are used to make two nodes for the bipartite graph. A function of a class is linked to an attribute. If the attribute of this class is accessing this function, the degree of cohesion can be calculated as

Max.{[Set of Functions*Set of Attributes] – [Set of Bipartite Graph],0}.......................................(1)

Max.{[? attributes*? functioins]-[? arcs in Bipartate Graph],0}..(2)

In AVISAR, during the repetitive process of generating the chromosome for the execution of the GA for an OOS, it requires an iterative process for the selection of one or more individual chromosomes. The execution of the test case selection occurs as explained below.

Let TS1 given as below be a chromosomal gene with 15 Test Cases that are chosen in a random order from the Initial population, provides a given test suite (TS1) which is as shown below.

TS1 = {T1, T2, T3, , T15}

T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
T12	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21
T22	T23	T24	T25	T26	T27	T28	T29	T30	T31	T32
T33	T34	T35								

The value of fitness function for the initial population of original Test Suite is given as below,

fvTS = 6.3532.

The above value of fitness function for the initial population has been calculated by addition of the weights of the whole test cases that are being related to TS.

The value for the Fitness Functions of TS1, fvTS1 = 2.553.

The initial probability for the crossover (Cp) of test gene is being chosen in a random manner and Cp = 0.4.

This value of the fitness function for a test gene is calculated by addition of the weights of whole set of test cases that are related to the chosen individual test genes during a genetic operation. The commencing test suite during the inception TS1, is as follows.

T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
T12	T13	T14	T15							

4.2 Complexity Calculator:

4.3 In AVISAR, during the OOD, it accepts the Number of Classes (NOC) as input to a OOSUT for calculating the level of complexity. Complexity can be calculated as when allocating weights to the OO classes:

Complexity level = {1/NOC)*? $^{n}{}_{i=0}$(Cw n(i) * NOM n(i))} ....(3)

Where,
Cw n(i) = Complexity weights of the n Classes,
NOM n(i) = Number of methods in individual n classes.

While calculating the effort estimation in AVISAR, our aim is to keep the level of complexity for an OOSUT, as lower as possible. The consecutive iterations for generation of test suites in AVISAR are given as below. The reasonable predictive numbers are generated for each test case gene of TS1 are listed below in Table 1.1. where Mutation Probability = 0.10.

R(Ti) for TS1			
T1	0.192	T9	0.333
T2	0.458	T10	0.458
T3	0.285	T11	0.088
T4	0.235	T12	0.411
T5	0.435	T13	0.723
T6	0.788	T14	0.052
T7	0.295	T15	0.482
T8	0.794		

Table 1.1 Arbitrary numbers generated during the mutation with probability is 0.10 on TS1 for its each test case

During the successive iterations for generating the test suites using the GA, The Test Cases for the individual genes test sets are being generated in a random fashion with the mutation probability (Mp = 0.10) were being restored by a set of new test genes from the native test suite (TS) that were not related to the TS1. The crucial tests T11 and T14 are substituted by new test cases from the initial population of the Test genes. These test genes are to be chosen in a random manner. The crucial tests T17 are being substituted with T16. These crucial tests cases are chosen in a random order, as they would be the next available new test genes from the actual lists of the crucial test gene set TS. Let us consider TS2 is to be the new test chromosomal gene that has been obtained as a result of the mutation and crossover operation on the test genes depending upon the value of their fitness function. The value of the fitness function of IP, fvTS is 6.3532 and value of the fitness function of TS2 which is given by fvTS2 is 2.4084.

After the first iteration in the test gene, the value of Cp is obtained as result of 2.4084 divide by 6.3532, which is equal to 0.3791, which is comparatively lesser than the value the initial crossover probability that is equal to 0.4. So as a result the crossover operation first iteration in the test gene is being used as accepted convention and the new individual test gene is accepted as contemplate for the next iteration.

In AVISAR, the efficacy of this test suite has been evaluated and examines to present them separately by examining the other different metrics like, (i) Test Suite Efficiency, (ii) Complexity Calculation using APFD value, and (iii) Fault Coverage. As a resultant, the freshly resultant chromosome, TS2 is as shown below.

T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T16
T12	T13	T17	T15							

For the next successive alteration in the test chromosomal genes, the chosen individual test gene is TS2, which was generated as a resultant in the previous successive alteration in the test chromosomal genes. Let probability for the mutation of the successive alteration in the test chromosomal genes is 0.10. The numbers generated randomly for each test chromosomal genes test cases are being shown as a list in the Table 1.2 given below.

R(Ti) for TS2			
T1	0.002	T9	0.402
T2	0.399	T10	0.091
T3	0.351	T163	0.181
T4	0.813	T12	0.525
T5	0.667	T13	0.782
T6	0.298	T17	0.453
T7	0.270	T15	0.610
T8	0.184		

Table 1.2 Arbitrary numbers generated during the mutation with probability is 0.10 on TS2 for its each test case

The Test cases T10 & T1 are being substituted with new test cases in the test chromosomal genes from its existing population by utilizing the random selection approach new test cases in the test chromosomal genes. These test cases are being substituted with T19 & T18, which would be available next to from its actual population, TS. The new test chromosomal genes are given as TS3, which will be attained after substituting these test genes that will meet the expectations of the convincing value of the fitness function.

4.4 OOD Estimator (OODE): In AVISAR, the OODE provides a general estimate of the various characteristics of OOD, such as inheritance depth encapsulation, visibility of methods (Vm) and degree of polymorphism for the OO classes. This **OODE** component in a SUT takes its NOC as input. Further we will examine its various subcomponents as given below:

4.3.1 Visibility Calculator: The total no: of methods that would be visible in the given classes in a OOSUT will be calculated by this. Let us consider Tc be the complete no: of classes in the OOSUT, Let us say Md(Ci) be the no: of methods that are being asserted in a Class. If a class in OOSUT accesses a method by calling the object of that class, the predicate of class in OOSUT can be seen; otherwise, it cannot be seen, so the Visibility for a Method in the OOSSUT can be found as:

$$Vm = \sum_{j=1}^{Tc} \frac{Pradicate\ of\ [Md(Ci)]i.e.Visible\ Classes}{Tc-1} \quad \ldots\ldots\ldots\ldots\ldots\ldots(4)$$

Where,

Vm = Visibility of Methods in a OOSUT's Class.

4.3.2 Encapsulation Measurement: The total number of classes Tc obtained from the NOC, in OOSUT is provided are feed as input to the **Encapsulation Measurement** component for OOSUT, provides the AHF and MHF for measuring the encapsulation for OOSUT, given as follows. The MHF and AHF jointly decide the encapsulation measurement of a class for OOSUT.

$$AHF = \sum_{i=1}^{Tc} \sum_{m=1}^{Ad} (1-Va) \,/\, \sum_{i=1}^{Tc} Ad \quad \ldots\ldots\ldots\ldots\ldots\ldots(6)$$

Where,

AHF = Attribute Hiding factor

Tc = Total number of classes

Ad = Attributes declared in a class in a SUT

Va = Visibility of attributes by a class in a SUT.

$$MHF = \frac{\sum Tci{=}1 \sum Mdm{=}1\,(1-Vm)}{\sum Tci{=}1\ Md} \quad \ldots\ldots\ldots\ldots\ldots\ldots(5)$$

Where,

MHF = Methods Hiding factor,

Tc= Total number of classes,

Md= Number of methods declared in the class in a SUT,

Vm= Visibility of methods invoked by a class in a SUT.

4.3.3 Inheritance Measurement: In an OOSUT the inheritance in the OO classes can be obtained as two different measures for examining the level of inheritance. The total number of classes which is denoted by Tc, will be taken as input for an OOSUT. It mathematically determines the number of methods that can be referred within the interconnected class in an OOSUT as Ma.

$$Ma = Md + Mi \quad \ldots\ldots\ldots\ldots\ldots\ldots (7)$$

Where,

Ma = Methods invoked in association with a Class.

Md = Methods declared in a Class

Mi = Methods inherited in a Class.

So the Method associated component Method Inheritance Factor (MIF) would determine the Methods in OOSUT to be inherited as given below:

$$MIF = \frac{\sum Tci{=}1\ Mi}{\sum Tci{=}1\ Ma} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(8)$$

Where,

MIF = Method Inheritance Factor,

For the Attribute Inheritance Factor (AIF), Attribute associated component would determine the number of attributes that can be approached in consortium with a OO class in a OOSUT as Aa.

$$Aa = Ad + Ai \quad \ldots\ldots\ldots\ldots\ldots\ldots (9)$$

Where,

Aa= Number of attributes that can be accessed in association with class i.

Ad = Number of attributes declared in a Class.

Ai = Number of attributes from a base Class that are accessible to a Class i which is inherited from a base Class

So the Attribute Inheritance Factor (AIF) will determine the attributes that would be approachable to a Class I, that can be derived genetically from a parental OO Class shown as:

$$AIF = \frac{\sum Tci{=}1\ Ai}{\sum Tci{=}1\ Ad} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(10)$$

Where,

AIF = Attribute Inheritance Factor.

4.3.4 Polymorphism Factor: The "polymorphism factor" (PF) in an OOSUT is to ascertain the size of the possibility to apply the polymorphism. The implementing prospective for the polymorphism factor in an OOSUT can be estimated as:

$$PF = \frac{Mo}{Mn * Dc} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(11)$$

Where,

Mo = Number of overriding methods in Class i.

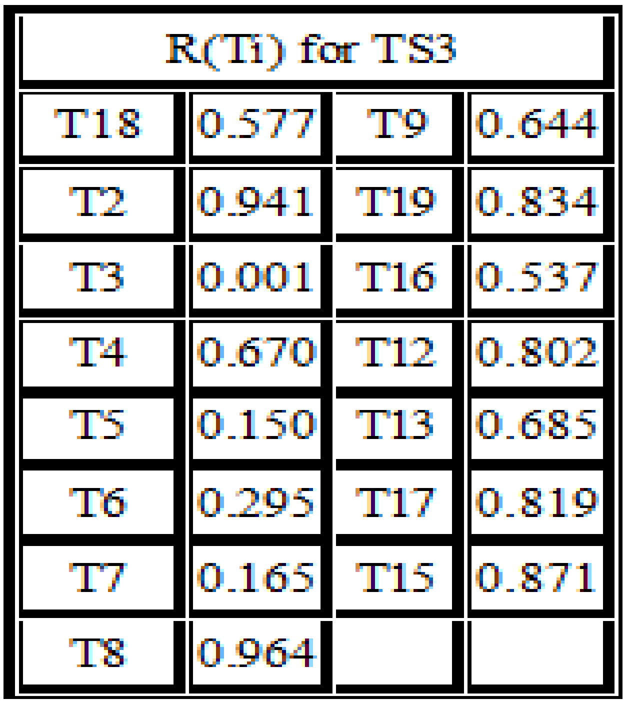
Mn = Number of new methods in Class i.

Dc = Number of descendents of Class i.

For calculating the Fitness value after the second successive alteration of IP, we have obtain the value of fvTS is to be equal to 6.3532.The fitness value of TS3 obtained during the second successive alteration of fvTS3 is to be equal to 2.3608. After the third successive alteration the Current value of the factor Cp can be calculated by dividing the fitness value of TS3 by value of fvTS which is found to be equal to 0.3716, which is less than the previous value of Cp which was equal to 0.3791. so as a result both the mutation and crossover operations that were accomplished, are fully approved as a new chromosome which is given as below.

T18	T2	T3	T4	T5	T6	T7	T8	T9	T19	T16
T12	T13	T17	T15							

This chromosomal gene TS3 is contemplated for next successive alteration. The probability for the new mutation process is found to be equal to 0.60. The successive random patterns of numbers are generated for the every test case of the chromosomal gene TS3 as shown below.

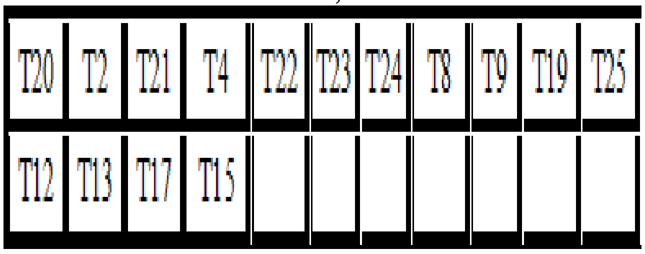


R(Ti) for TS3			
T18	0.577	T9	0.644
T2	0.941	T19	0.834
T3	0.001	T16	0.537
T4	0.670	T12	0.802
T5	0.150	T13	0.685
T6	0.295	T17	0.819
T7	0.165	T15	0.871
T8	0.964		

Table 1.3 Arbitrary numbers generated during the mutation with probability is 0.60 on TS3 for its each test case

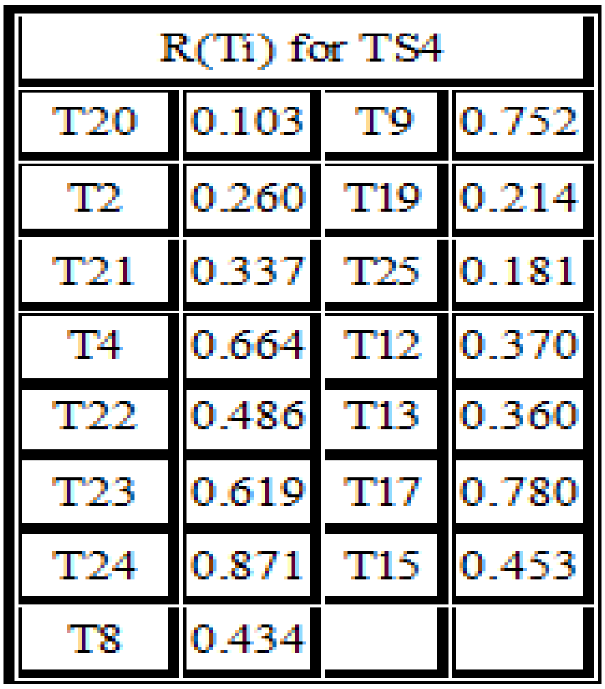
During the third crossover the chromosomal test genes that tend to fail to meet the expected mutation probability were substituted. So these test cases T18, T3, T5, T6, T7, T16 are being substituted in accordance with the arbitrary test cases as T20, T21, T22,T23,T24, T25.

Suppose the new chromosomal test genes is to be TS4, which is a resultant of substitution of the chromosomal test genes. The Fitness value of IP is given be the fitness value (fvTS) which is found to be equal to 6.3532 and the fitness value (fvTS4) of the chromosomal test gene TS4 is found to be equal to 2.3283. After the third successive iteration, the value of Cp is obtained by dividing fvTS by fvTS4 and their value is found to be equal to 0.3665, which is less than the fitness value of the TS3 which is 0.3716. Therefore, after the successive genetic crossover and genetic mutation operations that are being on the Chromosomal test gene TS3 & TS4 the new resultant chromosome TS4, is shown as follows.



T20	T2	T21	T4	T22	T23	T24	T8	T9	T19	T25
T12	T13	T17	T15							

Let us further examine the Chromosomal test suite for the test gene TS4 for the next successive alteration. Here the probability of occurrence of the mutations on chromosomal test gene is fixed at the value equal to 0.50. During this successive alteration arbitrary numbers were generated during the mutation on the chromosomal test gene TS4 for its each test case.

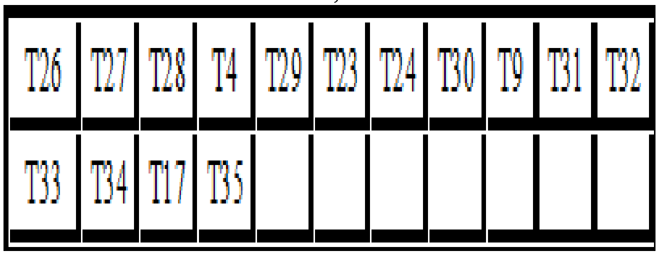


R(Ti) for TS4			
T20	0.103	T9	0.752
T2	0.260	T19	0.214
T21	0.337	T25	0.181
T4	0.664	T12	0.370
T22	0.486	T13	0.360
T23	0.619	T17	0.780
T24	0.871	T15	0.453
T8	0.434		

Table 1.4 Arbitrary numbers generated during the mutation with probability is 0.50 on TS4 for its each test case

During the fourth crossover the chromosomal test genes that tend to fail to meet the expected mutation probability were substituted. So these test cases T20, T2,T21, T22, T8, T19, T25, T12, T13, T15 are being substituted in accordance with the arbitrary test cases as T26, T27, T28,T29,T30, T31,T32, T33, T34, T35.

Suppose the new chromosomal test genes is to be TS5, which is a resultant of substitution of the chromosomal test genes. The Fitness value of IP is given be the fitness value (fvTS5) which is found to be equal to 6.3532 and the fitness value (fvTS5) of the chromosomal test gene TS5 is found to be equal to 2.9123. After the third successive iteration, the value of Cp is obtained by dividing fvTS by fvTS5 and their value is found to be equal to 0.4584, which is not less than the fitness value of the TS3 which is 0.3716. Therefore, after the successive genetic crossover and genetic mutation operations that are being on the Chromosomal test gene TS3 & TS4 the new resultant chromosome TS5, is shown as follows.



T26	T27	T28	T4	T29	T23	T24	T30	T9	T31	T32
T33	T34	T17	T35							

After the second stage crossover and further mutations on genes using the genetic algorithms the resultant analysis of the computed details and results attained for the AVISAR framework are being encapsulated and being shown in Table 1.5. Further Table 1.6 provides Analysis of data and results for the various test suites using the GA.

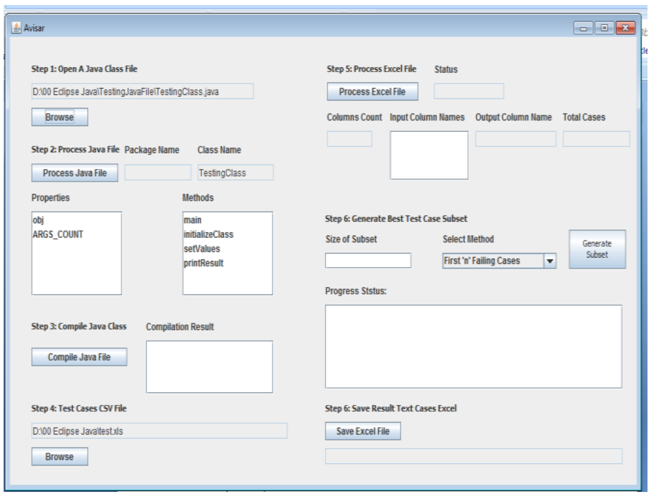
Test Suites & Information	IP-TS	TS1	TS2	TS3	TS4	TS5
Test Suite Size	35	15	15	15	15	15
# of faults not covered	0	5	4	5	2	0
# of TC revealing at least a new fault	12	7	8	7	8	11
# of Requirements Covered	10	5	7	6	6	10
Are all faults exposed?	Yes	No	No	No	No	Yes
METRICS						
APFD Value (%)	66.4	81	73	74	70	55.9
Requirements Coverage (%)	100	50	70	60	60	100
Fault Coverage (%)	100	58.3	66.7	58.3	83.3	100
Test suite Efficiency (%)	34.3	46.7	53.3	46.7	53.3	73.3

Table 1.5 Summed up results produced by genetic GA for OOP

V. EXPERIMENTAL SETUP

For effort estimation during the functioning AVISAR framework we have used the following code block for ensuring the max code coverage.

```
public GAMaxCodeCoverageCases
( File jF, List<String[]> ea, int mini, int maxi, int sos )
{       selectedJavaFile = jF;
        excelArray = ea;
        minIndex = mini;
        maxIndex = maxi;
        sizeOfSubset = sos; }
    public List<Integer> startGA()
{int i;
    IntegerChromosome ich      IntegerGene ig;       Integer
caseNumber;
    Genotype.of( BitChromosome.of(10,0.5) );
final Factory<Genotype<IntegerGene>> gtf =
Genotype.of(IntegerChromosome.of( minIndex, maxIndex,
sizeOfSubset ));
final Engine<IntegerGene, Integer>
engine=Engine.builder(GAMaxCodeCoverageCases::eval,gt
f).build();
final Genotype<IntegerGene> result =
engine.stream().limit(10).collect(EvolutionResult.toBestGen
otype());
System.out.println("Result : \n\t" + result);
System.out.println( "Count of failed cases in Result are: " +
String.valueOf(eval(result)) );
List<Integer> subsetCases = new ArrayList<Integer>();
        ich =
result.getChromosome().as(IntegerChromosome.class);
for( i=0; i<=(ich.length()-1); i++ )
{ig = ich.getGene(i);
caseNumber = ig.intValue();
subsetCases.add(caseNumber);        }
        return subsetCases;     }
```



D:\00 Eclipse Java\TestingJavaFile\TestingClass.java

TestingClass

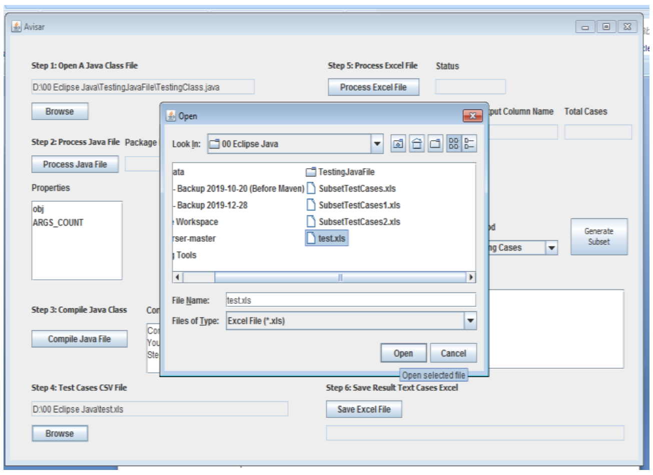
obj
ARGS_COUNT

main
initializeClass
setValues
printResult

D:\00 Eclipse Java\test.xls

First 'n' Failing Cases

Fig. 5.1. To show the interface of the tool AVISAR for implementation of the proposed work.

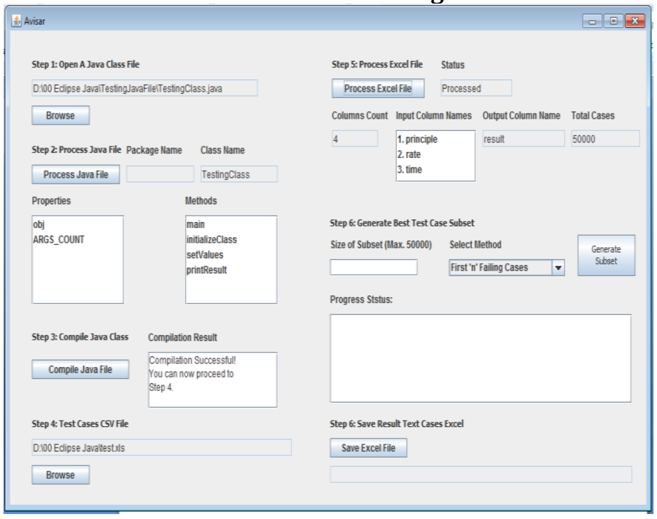


00 Eclipse Java

ata
Backup 2019-10-20 (Before Maven)
Backup 2019-12-28
Workspace
ser-master
Tools
TestingJavaFile
SubsetTestCases.xls
SubsetTestCases1.xls
SubsetTestCases2.xls
test.xls

File Name: test.xls
Files of Type: Excel File (*.xls)

Fig. 5.2. To show the selection of any Object oriented Source code to be tested using AVISAR



D:\00 Eclipse Java\TestingJavaFile\TestingClass.java

TestingClass

obj
ARGS_COUNT

main
initializeClass
setValues
printResult

Compilation Successful!
You can now proceed to Step 4.

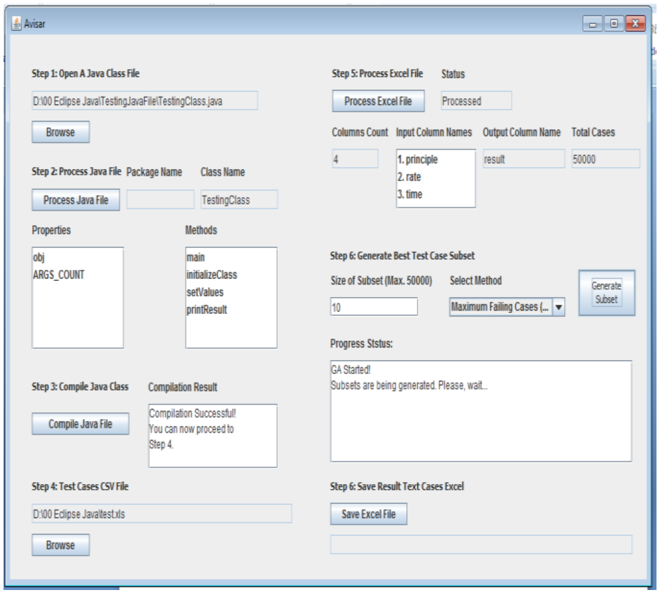
D:\00 Eclipse Java\test.xls

Status: Processed

Columns Count: 4
Input Column Names: 1. principle 2. rate 3. time
Output Column Name: result
Total Cases: 50000

First 'n' Failing Cases

Fig. 5.3.To show the execution of test cases using Genetic Algorithm



Avisar

Step 1: Open A Java Class File
D:\00 Eclipse Java\TestingJavaFile\TestingClass.java
Browse

Step 2: Process Java File | Package Name | Class Name
Process Java File | | TestingClass

Properties: obj, ARGS_COUNT
Methods: main, initializeClass, setValues, printResult

Step 3: Compile Java Class | Compilation Result
Compile Java File
Compilation Successful! You can now proceed to Step 4.

Step 4: Test Cases CSV File
D:\00 Eclipse Java\test.xls
Browse

Step 5: Process Excel File | Status
Process Excel File | Processed

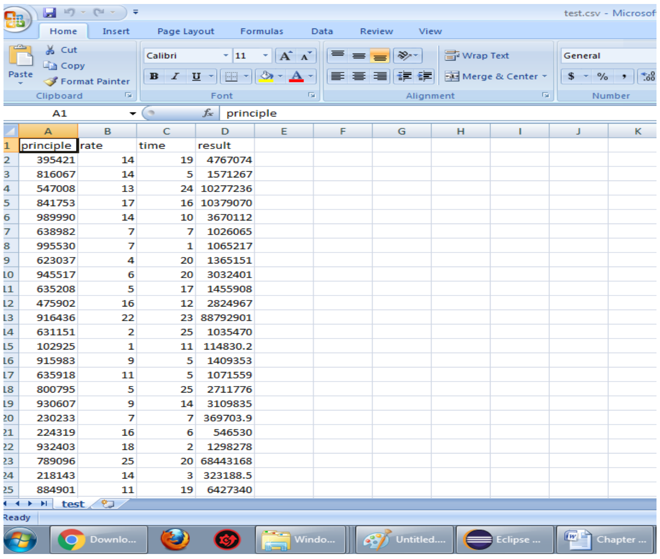
Columns Count	Input Column Names	Output Column Name	Total Cases
4	1. principle 2. rate 3. time	result	50000

Step 6: Generate Best Test Case Subset
Size of Subset (Max. 50000): 10
Select Method: Maximum Failing Cases (...
Generate Subset

Progress Ststus:
GA Started!
Subsets are being generated. Please, wait...

Step 6: Save Result Text Cases Excel
Save Excel File

Fig. 5.4. To show the immediate optimized result of test cases using the GA.



test.csv - Microsof

A1 principle

	A	B	C	D
1	principle	rate	time	result
2	395421	14	19	4767074
3	816067	14	5	1571267
4	547008	13	24	10277236
5	841753	17	16	10379070
6	989990	14	10	3670112
7	638982	7	7	1026065
8	995530	7	1	1065217
9	623037	4	20	1365151
10	945517	6	20	3032401
11	635208	5	17	1455908
12	475902	16	12	2824967
13	916436	22	23	88792901
14	631151	2	25	1035470
15	102925	1	11	114830.2
16	915983	9	5	1409353
17	635918	11	5	1071559
18	800795	5	25	2711776
19	930607	9	14	3109835
20	230233	7	7	369703.9
21	224319	16	6	546530
22	932403	18	2	1298278
23	789096	25	20	68443168
24	218143	14	3	323188.5
25	884901	11	19	6427340

test

Fig.5.5. To show the Test cases being designed for OOP under Test.

VI. EXPERIMENTAL RESULTS

When implemented in Java, the proposed OO Testing Framework AVISAR has provided better results for the small chunks of OO codes. We show the results as follows.

6.1 To measure cohesion, we achieved the following graph in Figure 6.1 when it was drawn between a set of functions and attributes together with the arcs in the bipartite graph. It showed impromptu results for smaller code, however as size of the code enlarges, it increases to infinity.

6.2 **To display the performance of Complexity Calculator** To measure the Complexity in Figure 6.2, we obtained the following graph, plotted between Number of classes and Number of methods in individual classes.

6.3 To show the overall performances of OOD estimator To measure the object-oriented design estimator, we plot the graph in Fig. 6.3 for the method visibility, Polymorphism, Inheritance, and Encapsulation for estimation of effort for smaller codes implemented using four modules in Java.

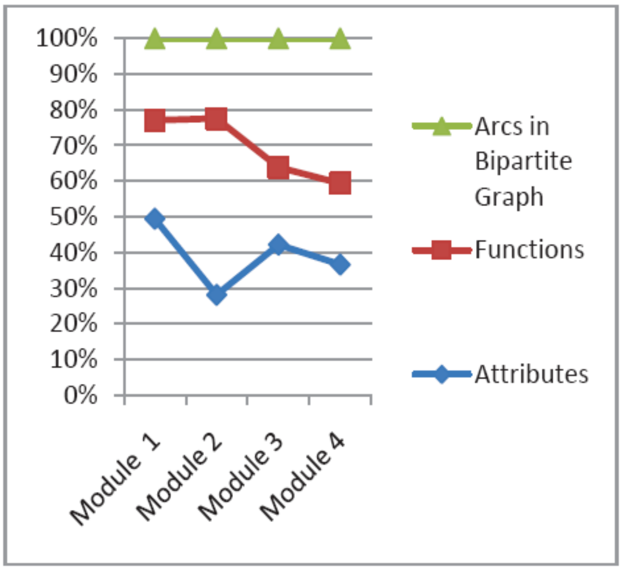


Fig. 6.1 Outturns for the Cohesion Estimator in AVISAR.

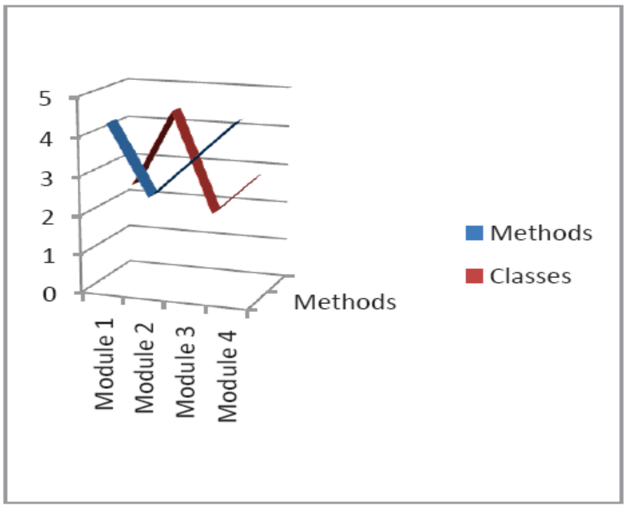


Fig 6.2 Outturns for the of Complexity Calculator in AVISAR.

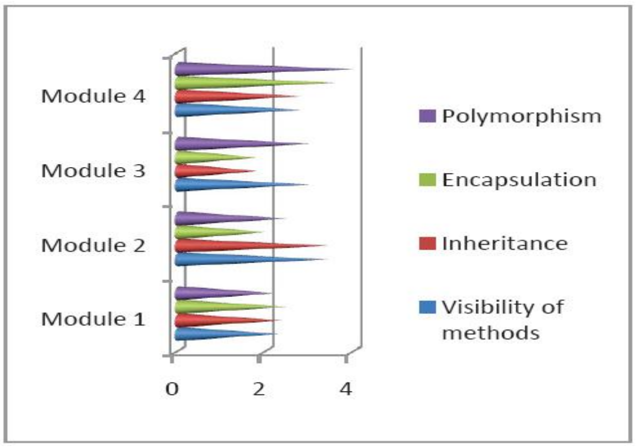


Fig. 6.3 Outturns for the of OOD estimator in AVISAR.

VII. CONCLUSIONS

In this research paper, we have proposed methodology for the evaluation of efforts putted in during the carrying out the trails of OOP for the detection of faults, using the AVISAR framework. During the evaluation of efforts using the AVISAR Framework we have observed that the use of GA has proven their worth to provide better results in terms of reduced effort estimation while testing the OOP.

Test Suite	Computation Steps	APFD Value (%)
TS	[1(1+3+2+4+7+12+11+17+21+23+26+20)/(12x35)] +[1/(2x35)] = 1-(147/420)+(1/70) = 1- 0.35 + 0.0143 = 0.6643	66.43
TS1	[1- (1+3+2+4+7+12+11+*+*+*+*+*)/(12x15)] + [1/(2x15)] = 1-(40/180)+(1/30) = 1 - 0.22 + 0.03 = 0.81	81.00
TS2	[1- (3+2+4+7+12+11+14+*+*+*+*)/(12x15)] + [1/(2x15)] = 1-(54/180)+(1/30) = 1- 0.3 + 0.03 = 0.73	73.00
TS3	[1- (*+3+1+4+7+12+11+14+*+*+*+*)/(12x15)] + [1/(2x15)] = 1-(52/180)+(1/30) = 1- 0.29 + 0.03 = 0.74	74.00
TS4	[1- (1+8+2+4+15+12+*+14+3+6+*+1)/(12x15)] + [1/(2x15)] = 1-(66/180)+(1/30) = 1- 0.37 + 0.03 = 0.70	70.00
TS5	[1- (3+5+9+4+8+10+5+14+7+6+1+13)/(12x15)] + [1/(2x15)] = 1-(85/180)+(1/30) = 1- 0.4722 + 0.03 = 0.6643	55.78

* refers to the fault not revealed by any of the test cases in the test suite.

Table 1.6 Analysis of data and results for the various test suites using the GA.

REFERENCES

1. Harumi W., Hiroki T. , Wenxin Wu , Motoshi Saeki; " A Technique for Analyzing and Testing Object-Oriented Software Using Colored Petri Nets"; IEEE International Conference;1998;Pg.No:182-190.
2. Aynur Abdurazik, Jeff Offutt;" Using Coupling-based Weights for the Class Integration and Test Order Problem" published by Oxford University Press, The British Computer Society, 2006.
3. A.S. Parrish, R.B. Borie, and D.W. Cordes, "Automated Flow Graph Based Testing of Object Oriented Software Modules," Journal of Systems and Software, 23, 1993, pp. 95-109.
4. Jaroenpiboonkit, J. , Suwannasart, T. ;" Finding a Test Order using Object-Oriented Slicing Technique" IEEESoftware Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific Pg. no: 49-56.
5. R.K. DOONG, P.G. Frankl; "The ASTOOT Approach to Testing Object Oriented Programs"; ACM Transactions on Software Engineering and Methodology, Vol. 3,1994, pages 101-130.
6. Wayne Liu And , Wayne Liu , Paul Dasiewicz;"The Event-Flow Technique for Selecting Test Cases for Object-Oriented Programs";IEEE conference proceedings 1997.
7. D.L. Parnas and Y. Wang, "Simulating the behavior of software modules by trace rewriting systems", IEEE Trans. Software Eng. 20 (10) (1993) Pg.No: 750-759.
8. Ilinca C, Andreas L., Manuel O., Bertrand M.;" ARTOO: Adaptive Random Testing for Object-oriented Software"; published in ICSE'08, May 10–18, 2008, Leipzig, Germany.
9. Yabo Wang, D. L. Parnas;" Simulating the Behavior of Software Modules by Trace Rewriting";IEEE transactions of software engineering;October1994(Vol.20,No.10) Pg. no.750-759.
10. H.Y. Chen, T.H. Tse, F.T. Chan, and T.Y. Chen., "In black and white: an integrated approach to class-level testing of object-oriented programs." ACM Transactions on Software Engineering and Methodology, 7(3):250–295, 1998.
11. W.K.Chan,T.Y.Chen,T.H.Tse;" 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).
12. Zhe Li, Hamilton M., T.;" An Approach to Integration Testing of Object Oriented Programs"; IEEE transactions, Quality Software, 2007. QSIC '07. Seventh International Conference, Oct. 2007; Pg. No: 268 – 273.
13. Alisa Irvine, A. Jefferson Offutt;" The Effectiveness of Category Partition Testing of Object-Oriented Software"; CiteseerX, 1995;
14. Tao Xie, David Notkin;" Automatically Identifying Special and Common Unit Tests Based on Inferred Statistical Algebraic Abstractions"; 2003.
15. S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, T. Anderson; "Eraser A dynamic data race detector for Multithreaded Programs"; ACM transactions on computer systems, 1997.
16. Gerald H., André B., Jan B.; "A distributed real-time java system based on csp"; CiteseerX, 2000.
17. Vats, P., "AVISAR - a three tier architectural framework for the testing of Object Oriented Programs" pub. In Second IEEE International Innovative Applications of Computational Intelligence on Power, Energy and Controls with their Impact on Humanity (CIPECH), 2016.

AUTHORS PROFILE

Prof. Manju Mandot is a Professor and Director of Directorate of Jan Shikshan and Extension, J.R.N. Rajasthan Vidyapith (Deemed University). She has 27 years of teaching experience. Her research interest includes image processing, E- governance, women empowerment with technology. She is esteemed member of Computer Society of India

Mr. Prashant Vats is working in the field of CSE & IT as an Assistant Professor from past 11 years. He has done Diploma in Medical Electronics, B. tech. (IT), M. Tech. (IT) from GGSIPU, New Delhi., MBA & M.A in Education from IGNOU, New Delhi, PG Diploma in Cyber Laws from Indian Law Institute, New Delhi. He is pursuing Ph.D. in CSE from Banasthali Vidyapith, Rajasthan. He is a member of IEEE.His research area includes OO paradigm, Software Engineering, IOT, Cloud Computing, BigData. He has published more than 32 research publications in various national & international journals of repute.