

# **Mutation Analysis Using Mutation Testing Tools by using Evosuite Generated Test Suites**

## **1 INTRODUCTION**

There are many coverage criterion used to check the code quality but Mutation testing has been proposed as stronger criterion because other traditional test coverage checks only how much code is executed by test suites and doesn't focus on the fault testing in the existing code. <sup>[1]</sup> Mutation Testing is one type of (white box) software testing where the source code is modified (mutated) and we check using the test cases if the results are erroneous or not. If results give us the error, then it is said that mutation is killed otherwise it is survived. This kind of unit testing is very expensive as we have to generate numerous mutated files and for each file we need to run the test cases to check if the bugs are identified.

The Scope of this project is to understand and analyze the mutation tools like PIT, MuJava, Major and implement it by using case study and Junit test cases generated by Evosuite.

The goal of this project is to understand mutation testing and implement it by using case study especially java project/files and multiple mutation testing tools. Evosuite can generate the Junit test cases from source files. Using these test cases, observe the results of mutation testing implemented on other mutation analysis tools and conclude the results.

## **2 STATE OF THE ART OVERVIEW**

EvoSuite automatically generates test suites that includes Junit test case for Java projects. <sup>[6]</sup> In this generated test suites, set of assertions are created to find the FOM (first order mutants). <sup>[2, 6]</sup> I have used Evosuite on command line for case study to produce the test suite and depending on that test suite, mutation score is analyzed by running all coverage criterion.

Apache Maven is used here with Evosuite to manage the project and integrate it with different mutation tools as it has large data in its repository. It makes easy build process and provides quality project information. By building project object model (POM.xml) in the project directory, Evosuite's dependencies and repositories can be added.

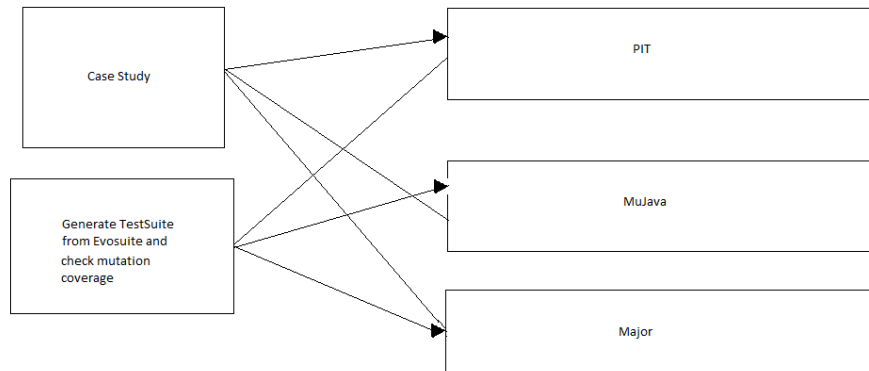
In this project we have used Evosuite, PIT, and MuJava to test the mutation by using the JUnit test cases. Another mutation tool, Major is left out in this project as the configuration was not done correctly.

PIT is a mutation testing system which is easy to use and can analyze the results efficiently. The results produced by PIT can be seen in one html file with name as "YYYYMMDDHHMM" in which it includes details about line coverage and mutation. PIT will focus on Line coverage and this data is collected first. PIT uses its own mutation engine.

Evosuite and PIT do not give the mutated source code (java and class files). MuJava (Mutation System for Java) is used to generate and export the mutated version of class with its source files in existing machine and test the mutation. It provides graphical user interface for every function. All JUnit test class should have public access to methods and it should start with string "test". <sup>[3]</sup>

### **3 DESIGN OF THE CASE STUDY**

Figure 1 illustrates how we are going to execute mutation testing.



**Figure 1 Process to be followed for Mutation Testing**

- This section will describe each case study file, mutation system tools, methodology and implementation.
- Case Study Files(Program Under Test):
  1. Ordset.java file: This is the main class which implements an ordered set of integers. It has different public and private methods and is used for different operations like removal, insertion and basic set operations. Public Methods includes operations:  
  
Equals(), contains(), remove(), add(), elementAt(), make\_a\_free\_slot(), union(), binSearch().  
  
Private Methods includes operations:  
  
updateLast(), defSetSize(), initSetArray(), resizeArray().
  2. OverflowException.java file: This file extends the Exception to throw an exception.
- These source files are kept in one Project and compiled using Maven on command line and then Evosuite generates the Junit test cases, which is the main part for study. Steps to compile source files and to generate Junit test suites are

given in “Steps To Generate Test Suite for OrdSet in Evosuite.docx”. Evosuite uses SimpleMutationAssertionGenerator to generate the mutators.

- Mutation operators has two types
  - (1) Method Level Operators: These operators are used mainly for unit testing<sup>[1,2,4]</sup>
  - (2)Class Level operators: This operator are used not only for unit but also for integration testing. <sup>[1,2,4]</sup> First Order Mutants (FOMs) can be created by only changing a single statement. For example change from if (x>y) to if(x<y). Higher Order Mutants (HOMs) can be created by mutating or changing multiple lines. <sup>[2]</sup> Here, we are only using the First Order Mutants for our mutation analysis and different operators to achieve this FOM.
- I have used the Evosuite generated Junit file on PIT to check the mutation. The reports will be generated under “target/pit-reports” which will give details about line and mutation coverage. We have only used mutationCoverage goal to evaluate the mutation to all 2 classes. The scmMutationCoverage goal is not covered as it only checks those classes that matches the source files and filters within the project. The instruction on how to generate mutation coverage is given in “Steps To Run TestSuite on PIT.docx”. PIT will use below mentioned mutators:
  - ConditionalsBoundaryMutator
  - IncrementsMutator
  - VoidMethodMutator
  - ReturnValsMutator
  - MathMutator
  - NegateConditionalsMutator.
- MuJava gives GUI representation and is user-friendly and easy to access. The steps on how to operate the MuJava is given in “Steps To Run TestSuite on MuJava.docx”. I have used the class and method level operators that are described in the Figure 2 as below :

Method Level operator	Class level Operator
-----------------------	----------------------

AORB	IHI
AORS	IHD
AOIU	IOD
AOIS	IOP
AODU	IOR
AODS	ISI
ROR	ISD
COR	IPC
COD	PNC
COI	PMD
SOR	PPD
LOR	PCI
LOI	PCC
LOD	PCD
ASRS	PRV
SDL	OMR
VDL	OMD
CDL	OAN

ODL	JTI
	JTD
	JSI
	JSD
	JID
	JDC
	EOA
	EOC
	EAM
	EMM

**Figure 2 Illustration of Mutation Operators Used By MuJava**

- The comparison of mutation testing is done based on the mutation score.

## 4 ANALYSIS OF THE RESULTS

- As per the Statistics.csv, the Weak Mutation score of OrdSet for Evosuite is 0.9402 which means 94.2% mutation killed out of 100%. The Evosuite gives Weak Mutation score for OverflowException 1, which means 100% mutations are killed but this is of no use as it only extends the Exception class and we are only interested in OrdSet class coverage. Line coverage for OrdSet is 0.9444 which is also a good result and highlighted in figure3.

	I	J	K	L	M	N	O	P	Q	R	S	T	W	
1	Size	Length	Total_Tim	Random	LineCoverage	LineCover	BranchCov	BranchCov	Exception	Exception	Explicit_M	Ex	WeakMutationScore	W
2	43	622	152806	2E+12	0.9444444444	1.1E+143	0.887755	1.11E+97	1	1111111	0	0	0.940217391	11
3														
4														
5														
6														

**Figure 3 Line Coverage and Weak Mutation Score of Evosuite Testing**

- PIT has line coverage of 99% which is better than Evosuite but mutation coverage is 0%. PIT generates mutation at runtime and doesn't store it anywhere on disk. The Detailed mutation generation count, Killed, Survived data for each mutation operator are mentioned in figure 4.

```
=====
- Mutators
=====

> org.pitest.mutationtest.engine.gregor.mutators.ConditionalsBoundaryMutator
>> Generated 34 Killed 0 (0%)
> KILLED 0 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 34

> org.pitest.mutationtest.engine.gregor.mutators.IncrementsMutator
>> Generated 7 Killed 0 (0%)
> KILLED 0 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 7

> org.pitest.mutationtest.engine.gregor.mutators.VoidMethodCallMutator
>> Generated 13 Killed 0 (0%)
> KILLED 0 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 13

> org.pitest.mutationtest.engine.gregor.mutators.ReturnValsMutator
>> Generated 33 Killed 0 (0%)
> KILLED 0 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 33

> org.pitest.mutationtest.engine.gregor.mutators.MathMutator
>> Generated 43 Killed 0 (0%)
> KILLED 0 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 43

> org.pitest.mutationtest.engine.gregor.mutators.NegateConditionalsMutator
>> Generated 45 Killed 0 (0%)
> KILLED 0 SURVIVED 0 TIMED_OUT 0 NON_VIABLE 0
> MEMORY_ERROR 0 NOT_STARTED 0 STARTED 0 RUN_ERROR 0
> NO_COVERAGE 45

=====

- Timings
=====

> scan classpath : < 1 second
```

Figure 4 Mutators generated at runtime for PIT Mutation testing

- PIT mutation test coverage is shown in figure 5 which shows that out of 144 lines, 143 lines are covered which gives 99% of Line coverage. It has generated total of 175 mutators in this case. When we click on the OrdSet.java, we are able to see

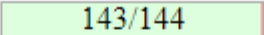
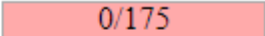


the code of source file. If the statement has dark green line then it means that mutation is killed and vice versa for light pink colored statement. Along with the details of active mutators, time taken by PIT to execute mutation testing is given at the end of the page. This illustration is shown in figure 6.

## Pit Test Coverage Report

### Package Summary

default

Number of Classes	Line Coverage	Mutation Coverage
1	99%  143/144	0%  0/175

### Breakdown by Class

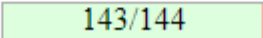
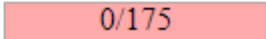
Name	Line Coverage	Mutation Coverage
<a href="#">OrdSet.java</a>	99%  143/144	0%  0/175

Figure 5 PIT Mutation Test Coverage

3.	replaced return of integer sized value with (x == 0 ? 1 : 0) → SURVIVED
1.	changed conditional boundary → SURVIVED
431	2. negated conditional → SURVIVED
3.	replaced return of integer sized value with (x == 0 ? 1 : 0) → SURVIVED
432	1. Replaced integer division with multiplication → SURVIVED
1.	Replaced integer multiplication with division → SURVIVED
433	2. negated conditional → SURVIVED
3.	replaced return of integer sized value with (x == 0 ? 1 : 0) → SURVIVED
1.	Replaced integer addition with subtraction → SURVIVED
434	2. Replaced integer multiplication with division → SURVIVED
3.	replaced return of integer sized value with (x == 0 ? 1 : 0) → SURVIVED
450	1. changed conditional boundary → SURVIVED
2.	negated conditional → SURVIVED
452	1. removed call to OrdSet::add → SURVIVED
454	1. removed call to OverflowException::printStackTrace → SURVIVED
465	1. Replaced integer addition with subtraction → SURVIVED
1.	changed conditional boundary → SURVIVED
467	2. changed conditional boundary → SURVIVED
3.	negated conditional → SURVIVED
4.	negated conditional → SURVIVED
1.	changed conditional boundary → SURVIVED
470	2. Changed increment from 1 to -1 → SURVIVED
3.	Replaced integer addition with subtraction → SURVIVED
4.	negated conditional → SURVIVED
476	1. Replaced integer addition with subtraction → SURVIVED

### Active mutators

- CONDITIONALS\_BOUNDARY\_MUTATOR
- INCREMENTS\_MUTATOR
- INVERT\_NEGS\_MUTATOR
- MATH\_MUTATOR
- NEGATE\_CONDITIONALS\_MUTATOR
- RETURN\_VALS\_MUTATOR
- VOID\_METHOD\_CALL\_MUTATOR

### Tests examined

- OrdSet\_ESTest.OrdSet\_ESTest (4156 ms)

Report generated by: PIT 1.4.10

**Figure 6 Details and status of Active mutators in OrdSet.java**

- MuJava generates 1540 traditional mutants and 28 class mutants. It doesn't check line coverage and it gives mutation score 0% in which only 4/1568 traditional mutants (AOIS58, AOIS59, ODL32, ROR43) are killed. The final result is given in figure 7.

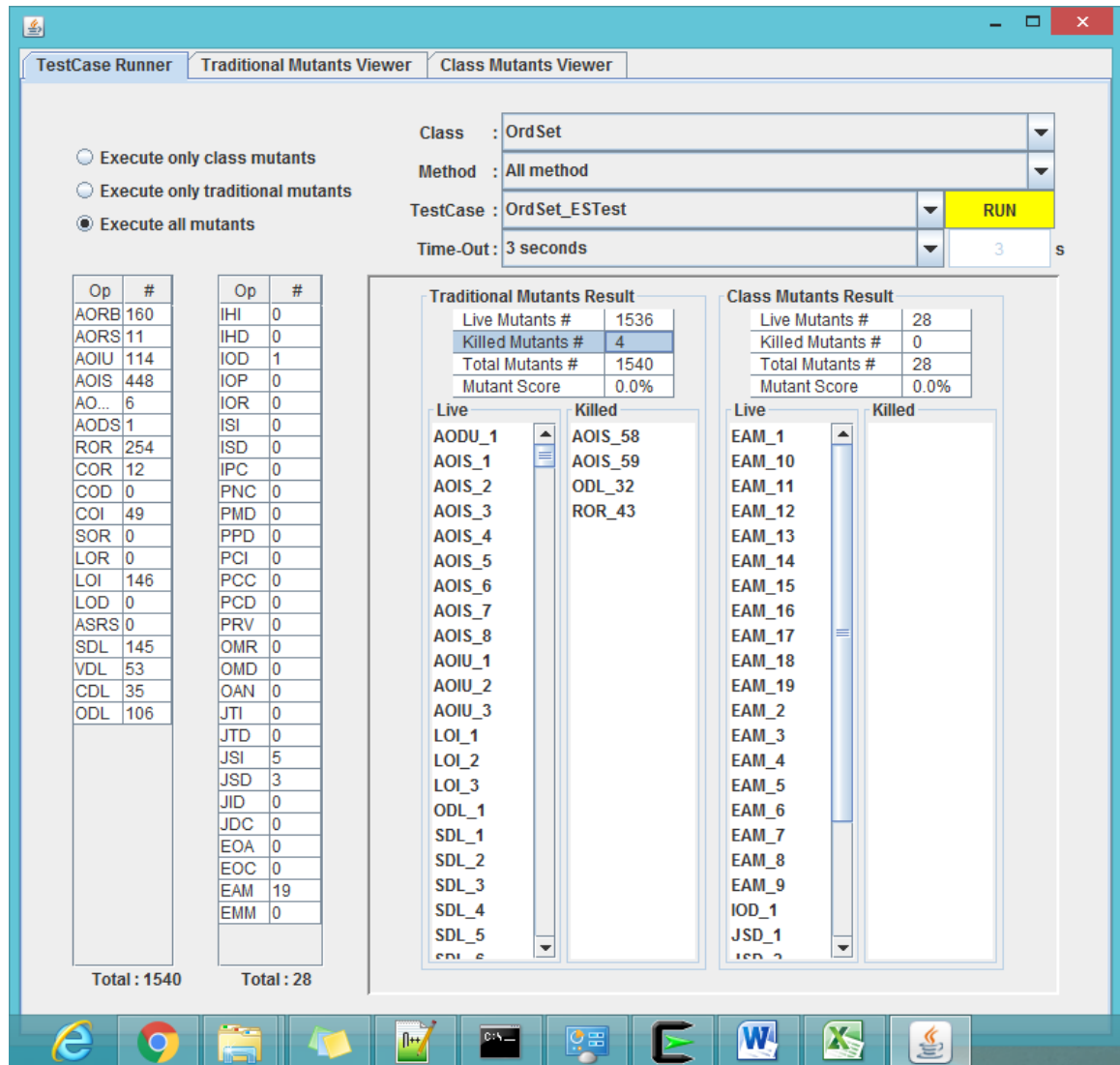


Figure 7 Mujava Mutation Testing Results

- The prediction for mutation score of other mutation software tools was above 80% but it is not fulfilled and there are many reasons why this failure happened and they are mentioned below:
  - There were many Out of Bound exception messages that appeared while testing the OrdSet file. Because the Evosuite generated file OrdSet expects input from length 3 to 9 for Ordered set while OrdSet\_ESTest.java file has multiple test case scenarios in which there are only few test cases which pass input within this range of [3,9]. So, mutation won't be visible as it is

going to throw an Exception. The Out of bound element exception is shown in figure 8.

```
<evosuite>.<evosuite><evosuite>>
<evosuite>.<evosuite><evosuite>>
OrdSet.add(OrdSet.java:222)
<evosuite>.<evosuite><evosuite>>
<evosuite>.<evosuite><evosuite>>
OrdSet.add(OrdSet.java:229)
<evosuite>.<evosuite><evosuite>>
<evosuite>.<evosuite><evosuite>>
OrdSet.add(OrdSet.java:222)
<evosuite>.<evosuite><evosuite>>
<evosuite>.<evosuite><evosuite>>
OrdSet.add(OrdSet.java:222)
<evosuite>.<evosuite><evosuite>>
{test16=pass, test15=pass, test14=pass, test13=pass, test12=pass, test11=pass, test10=pass, test54=pass, test53=pass, test19=pass, test18=pass, test17=pass, test52=pass, test51=pass, test50=pass, test05=pass, test49=pass, test48=pass, test04=pass, test03=pass, test47=pass, test02=pass, test46=pass, test01=pass, test45=pass, test00=pass, test44=pass, test43=pass, test42=pass, test09=pass, test08=pass, test07=pass, test06=pass, test41=pass, test40=pass, test38=pass, test37=pass, test36=pass, test35=pass, test34=pass, test33=pass, test32=pass, test31=pass, test39=pass, test30=pass, test27=pass, test26=pass, test25=pass, test24=pass, test23=pass, test22=pass, test21=pass, test20=pass, test29=pass, test28=pass}
JSI_50Out of bound element: -2855
Out of bound element: -3080
Out of bound element: 1120
Out of bound element: 752
Out of bound element: 0
Out of bound element: 9
Out of bound element: 535
Out of bound element: 535
Out of bound element: -3036
Out of bound element: 9
Out of bound element: 1297
Out of bound element: -4936
Out of bound element: 2
Out of bound element: -4936
Out of bound element: 0
Out of bound element: 705
Out of bound element: 3
Out of bound element: 0
Out of bound element: 9
Out of bound element: -2233
Out of bound element: -2256
OrdSet.add(OrdSet.java:229)
<evosuite>.<evosuite><evosuite>>
<evosuite>.<evosuite><evosuite>>
OrdSet.add(OrdSet.java:222)
<evosuite>.<evosuite><evosuite>>
<evosuite>.<evosuite><evosuite>>
OrdSet.add(OrdSet.java:229)
<evosuite>.<evosuite><evosuite>>
<evosuite>.<evosuite><evosuite>>
OrdSet.add(OrdSet.java:222)
<evosuite>.<evosuite><evosuite>>
<evosuite>.<evosuite><evosuite>>
OrdSet.add(OrdSet.java:222)
{test16=pass, test15=pass, test14=pass, test13=pass, test12=pass, test11=pass, test10=pass, test54=pass, test53=pass, test19=pass, test18=pass, test17=pass, test52=pass, test51=pass, test50=pass, test05=pass, test49=pass, test48=pass, test04=pass, test03=pass, test47=pass, test02=pass, test46=pass, test01=pass, test45=
```

Figure 8 Out Of Bound Element Exception in MuJava

- The test suite that is generated doesn't have assert statement which is going to pass each and every statement (shown in figure 8) and every mutant generated will not be able to give failure as all the results are going to pass. The reason for not inserting assert statement is found by checking the logs in which it is mentioned that the assertion doesn't happen because

SimpleMutationGenerator.java file reached maximum time to generate assertion and aborted the operation of assertion. The log files are under MAIN\_PROJECT\evosuite\tmp\_2019\_12\_05\_21\_03\_08\logs\OrdSet\std\_out\_CLIENT.log. As a result of this, PIT and MuJava were able to pass each and every statement and all tests were executed and 0% mutation score achieved. The 4 mutators that are passed in MuJava were able to kill mutation as the above mentioned out of bound exception was not thrown for that particular method.

- To conclude, Evosuite is able to kill First Order Mutants with mutation score of 94% while PIT and MuJava were not able to kill mutants as the Junit file generated by Evosuite is not qualitative for other tools and it needs modification which is out of scope for this project.

## 5 LESSONS LEARNED AND OPEN ISSUES

There were many technical issues faced which are illustrated as below:

- Naming Convention for the MAIN\_PROJECT folder as Maven was not able to compile the OrdSet.java file for the directory MAIN PROJECT.
- For MuJava testing, we have to additionally add the jar dependency into the CLASSPATH variable as it will not be able to find the Evosuite and Junit dependency. For this problem, I have modified classpath as:
  - “%CLASSPATH%;C:\MUJAVA\mujava.jar;C:\ProgramFiles\Java\jdk1.8.0\_231\lib\tools.jar;C:\MUJAVA\openjava.jar;C:\MUJAVA\classes;C:\MUJAVA\hamcrest-core-1.3.jar;C:\MUJAVA\junit-4.12.jar;C:\MUJAVA\evosuite-standalone-runtime-1.0.6.jar”;
- The standard structure in MuJava is to place Junit test files under C:\MUJAVA\testset directory. But it was not able to find the test file even if I put all files correctly in testset folder. For example the error was shown as “initializationError=initializationError: ; Class 'OrdSet\_ESTest.class' should be in target project, but could not be found!}”.

So as a resolution I found out that these 4 class files (OrdSet\_ESTest.class, OrdSet\_ESTest\_scaffolding.class, OverflowException\_ESTest.class, OverflowException\_ESTest\_scaffolding.class) should also be placed parallel to MUJAVA folder structure (For Example: C:\MUJAVA\).

- As it was strictly mentioned that I have to use the test case generated by Evosuite, I didn't try to change the generated Junit java file. I would have tried to insert assert statements in Evosuite generated test case if I had more time on this project as it takes time to understand each and every software along with its mutation mechanism.

The Suggestions to expand this project are illustrated below:

- The solution to the above mentioned concern will be to make 2/3 simple Junit tests for OrdSet.java file and 5-10 method and class level mutation operators to create mutants. This is easier to compare, as we can plot graph easily while in our case we cannot plot a graph if we don't have multiple Junit classes.
- Another Solution is to manually check the Junit tests generated by Evosuite and detect if any faults are there. This process will generate the test set that will give best results and number of tests will be minimal.<sup>[1]</sup>

## 6 REFERENCES

[1] Achieving scalable mutation-based generation of whole test suites. *Empirical Software Engineering*, (2015), Fraser, G., & Arcuri, A., pages 783-812.

[2] A Study on EvoSuite as an Automatic Test Case Generation Approach to Kill First Order Mutants, (2018), Homayouni, H., & Ghosh, S, pages 1-2

[3] MuJava: a mutation system for Java. In *Proceedings of the 28th international conference on Software engineering* Ma, (2006), Y. S., Offutt, J., & Kwon, Y. R., pages 12-13

[4] MuJava: A mutation system for java. In *Proceedings of the 28th International Conference on Software Engineering*, (2006), Yu-Seung Ma, Jeff Offutt, and Yong-Rae Kwon., pages 827-830

[5]Subtle higher order mutants. Information and Software Technology.(2017)  
Elmahdi Omar, Sudipto Ghosh, and Darrell Whitley.pages1-10

[6]EvoSuite: automatic test suite generation for object-oriented software., (2011),.  
Fraser, G., & Arcuri, A. Fraser, G., & Arcuri, A.. pages. 416-419

## **Appendix A   Appendices**

The source file OrdSet.java and OverflowException.java are in the package that is given along with this report. The folder will have all the documentation on instructions to configure Evosuite, Maven, PIT, MuJava software. If someone wants to replicate my work, there are multiple tutorial documents created to achieve those test results. All files related to Evosuite, Maven and PIT are in MAIN\_PROJECT folder while MuJava files will be under MUJAVA folder.