Software testing verification and validation

MIDTERM REPORT

Moustapha Sy Dieng
mxd152830

Ponmalar Silambaram Chandrabose
pxs162030

Trung Hieu Tran
txt171930

1. The problem statement

In 1970, Edsger W. Dijkstra said that "program testing can be used to show the presence of bugs, but never to show their absence" [1].

In fact, it can be easily demonstrated that even achieving a hundred percent code coverage does not guarantee bug-free software. For this reason, we can't effectively rely on code coverage as a metric for software quality. Our aim is to therefore utilize mutation testing to measure unit tests fiability. Mutation testing provides the ability to not only detect if code is executed but also evaluate the test suite ability to uncover regressions. It accomplishes this by introducing changes to the code, called mutations, and running the test suite against the newly modified code. In the next section, we will discuss some techniques used to implement it.

2. The basics of existing techniques and/or the design of your technique

2.1 Overview of PIT Mutation Testing:

Mutation testing is a concept where faults (or mutations) are automatically seeded into the code, tests are run and if any of the tests fail, then the mutation is killed. Otherwise the mutation lived. The percentage of mutations gauge the quality of the tests performed.

Specifically PIT Mutation Testing, an open source mutation testing tool is selected. Here, PIT runs your unit test against versions of application code that have been automatically modified. When the tests are run, since the application code has been changed it should produce different results. If a unit test does not fail in this situation, then there is some problem in the test suite [2].

2.2 Why use PIT?:

PIT Mutation Testing is fast because it operates on bytecodes and optimizes the executions of mutants. It can also be integrated into a wide range of development tools and also invoked through command line interfaces like Ant or Maven.

2.3 Existing Mutators:

Some of the mutators that are already available in the PIT Mutation Testing include

Conditionals Boundary Mutator : They replace the relational operators

Increments Mutator : They mutate the increments, decrements and assignment increments and decrements of local variables

Math Mutator : Replaces binary arithmetic operations for either integer or floating - point arithmetic

Invert Negatives Mutator : It inverts negation of integer and floating point numbers.

Return values mutator : It mutates the return values of the method calls depending on the type

Void Method calls Mutator : It removes method calls to void methods

3. Your implementation/study plan

Project Goals

The goal of the project was to augment the PIT mutation testing tool by adding three additional mutators:

1. AOD (Arithmetic Operator Deletion): Replace an arithmetic expression by each one of the operand.
2. AOR (Arithmetic Operator Replacement): Replace an arithmetic expression by each of the other ones.
3. ROR (Relational Operator Replacement): Replace the relational operators with each of the other ones.

Preparations

Our first task was to setup our environment. We opted to use the following configuration:

- IDE: Eclipse Oxygen
- Java version: JDK 1.8
- Build system: Apache Maven 3.5.3
- Version control: GitHub

Once we setup the environment, we created a small program along with a couple of test cases to ensure we were able to run unit tests properly. (Figure 1)

Figure 1

Next, we cloned the pitest repository from Github. We ran into our first obstacle when attempting to run mutation testing on our program. We were mistakenly attempting to build the project we created with the command 'mvn clean install' command. After much troubleshooting, we figured out that the pitest project needed to be built in this manner. After successfully building the project, we were able to run the default pitest mutators on our program successfully.

Phase 1:

Strategy

We decided to divide up the workload. Trung worked on implementing AOD, Ponmalar Silambaram Chandrabose took on AOR's implementation and Moustapha handled ROR's.

AOD

AOD replaces an arithmetic expression by each one of the operand. Below is the replacement table which is applied in the scope of the program.

Operator	Expression sample	Replaced by the first operand	Replaced by the second operand
+	a + b	a	b
-	a - b		
*	a * b		
/	a / b		
%	a % b		

First operand replacement

To replace an expression by the first operand, we had to remove the first parameter from the stack of java virtual machine which is associated the second operand in the expression. However, we needed to take into account the way JVM stores the different data types. Data types with sizes betweem 8-bits and 32-bits occupy 1 stack slot, while 64-bits data occupies 2 stack slots. Therefore, it would take 1 stack slot for *int* and *float* and this figure is 2 stack slots for *double* and *long*. Thus, we used the operand stack *POP* to remove item from the stack for *int* and *float* operators and the *POP2* for *double* and *long* operators. The code snippet in figure 3 shows the example of implementation for addition operator

Second operand replacement

To replace an expression by the second operand, there are 2 scenarios. For the int and float operators, we use operand *SWAP* to swap two items on the stack. Then, the order of operand in the expression is changed and the problem becomes the replace the expression by the first operand. In case of long and double operators, there doesn't exist any operand to swap first two slots with next two slots in the stack. So that we use *DUP2_X2* to duplicate the first two slots in the stack and push them after the 4-th slot in the stack. Then we use *POP2* two times to pop first 4-slots on the top of the stack and the remainder is the second operand of the expression. Below is the code snippet for those above scenarios.

```
if (this.context.shouldMutate(muID)) {
  if (type == 1) {
     super.visitInsn(Opcodes.SWAP);
     super.visitInsn(Opcodes.POP);
  } else {
     super.visitInsn(Opcodes.DUP2_X2);
     super.visitInsn(Opcodes.POP2);
     super.visitInsn(Opcodes.POP2);
  }
```

Figure 2

AOR

Definition

Arithmetic Operator Replacement is defined as the mutation process where one specific operator in an expression is replaced by all other operators.
For example: Let us consider a+b as an expression. This mutation operator should replace the '+' operator by subtraction, multiplication, division and modulus operator.

We also took into consideration the combinations of operators that can be replaced.
We used java byte code manipulation to replace these operators directly in the byte code of the program rather than change them manually in the system.

```
MUTATIONS.put(Opcodes.IADD, new InsnSubstitution(Opcodes.POP,
        "AOD Mutator: Removed the second operator from an addition formula (int)"));
MUTATIONS.put(Opcodes.DADD, new InsnSubstitution(Opcodes.POP2,
        "AOD Mutator: Removed the second operator from an addition formula (double)"));
MUTATIONS.put(Opcodes.FADD, new InsnSubstitution(Opcodes.POP,
        "AOD Mutator: Removed the second operator from an addition formula (float)"));
MUTATIONS.put(Opcodes.LADD, new InsnSubstitution(Opcodes.POP2,
        "AOD Mutator: Removed the second operator from an addition formula (long)"));
```

Figure 3

```
MUTATIONS.put(Opcodes.IADD, new InsnSubstitution(Opcodes.ISUB,"Replaced integer addition with subtraction"));
MUTATIONS.put(Opcodes.ISUB, new InsnSubstitution(Opcodes.IADD,"Replaced integer subtraction with addition"));
MUTATIONS.put(Opcodes.IMUL, new InsnSubstitution(Opcodes.IDIV,"Replaced integer multiplication with division"));
MUTATIONS.put(Opcodes.IDIV, new InsnSubstitution(Opcodes.IMUL,"Replaced integer division with multiplication"));
MUTATIONS.put(Opcodes.IOR, new InsnSubstitution(Opcodes.IAND,"Replaced bitwise OR with AND"));
MUTATIONS.put(Opcodes.IAND, new InsnSubstitution(Opcodes.IOR,"Replaced bitwise AND with OR"));
MUTATIONS.put(Opcodes.IREM, new InsnSubstitution(Opcodes.IMUL,"Replaced integer modulus with multiplication"));
MUTATIONS.put(Opcodes.IXOR, new InsnSubstitution(Opcodes.IAND,"Replaced XOR with AND"));
MUTATIONS.put(Opcodes.ISHL, new InsnSubstitution(Opcodes.ISHR,"Replaced Shift Left with Shift Right"));
MUTATIONS.put(Opcodes.ISHR, new InsnSubstitution(Opcodes.ISHL,"Replaced Shift Right with Shift Left"));
MUTATIONS.put(Opcodes.IUSHR, new InsnSubstitution(Opcodes.ISHL,"Replaced Unsigned Shift Right with Shift Left"));
```

Figure 4

The possible combinations are:

+	-	*	/	%
-	*	/	%	+
*	/	%	+	-
/	%	+	-	*
%	+	-	*	/

Thought Process
Solution 1:
We need to implement java byte code manipulation to replace these arithmetic operators in the byte code of the program directly.
We also need to take into consideration the various combinations of operator exchange possible in the system.
Solution 2: Manually go through the program and replace all of them, which becomes tedious if the program to be tested is really long.
Roadblocks:
Learning how to access the Java Byte code for that particular method.
Solution/Implementation:
We took into consideration the data types such as integer, long, double and float for arithmetic operators because each of them are stored differently in the stack.

We used the MethodVisitor function to look through the system and visit every method in the program.
In each method we look for the ADD, SUB, MUL, DIV, REM, as well as bitwise operation functions for each respective data type represented in the byte code and replace them with each other using the put function in java
We maintain HashMap values of each of these and create separate functions for each combination possible to make it faster to implement the test function.

ROR

The implementation of the relational operator replacement required for each relational operator to be replaced by every other one. The table below shows the all the possible replacements we implemented (note: We only handled the cases listed on the project guideline):

==	!=	>=	>	<=	<
!=	>=	>	<=	<	==
>=	>	<=	<	==	!=
>	<=	<	==	!=	>=
<=	<	==	!=	>=	>
<	==	!=	>=	>	<=

Solution

```
static {
    // Not Equal with Greater Than
    MUTATIONS.put(Opcodes.IF_ICMPEQ, new Substitution(Opcodes.IF_ICMPLE, "ROR Mutator: Replaced '!=' with '>'."));

    // Equal with Greater Than
    MUTATIONS.put(Opcodes.IF_ICMPNE, new Substitution(Opcodes.IF_ICMPLE, "ROR Mutator: Replaced '==' with '>'."));

    // Less Than with Greater Than
    MUTATIONS.put(Opcodes.IF_ICMPGE, new Substitution(Opcodes.IF_ICMPLE, "ROR Mutator: Replaced '<' with '>'."));

    // Less Than or Equal with Greater Than
    MUTATIONS.put(Opcodes.IF_ICMPGT, new Substitution(Opcodes.IF_ICMPLE, "ROR Mutator: Replaced '<=' with '>'."));

    // Greater Than or Equal with Greater Than
    MUTATIONS.put(Opcodes.IF_ICMPLT, new Substitution(Opcodes.IF_ICMPLE, "ROR Mutator: Replaced '>=' with '>'."));
}
```

Figure 5

```
package edu.utdallas.main;

public class Compare {

    public boolean compare(int a, int b) {
        return a > b;
    }
}
```

```
Compiled from "Compare.java"
public class edu.utdallas.main.Compare {
  public edu.utdallas.main.Compare();
    Code:
       0: aload_0
       1: invokespecial #1                  // Method java/lang/Object."<init>":()V
       4: return

  public boolean compare(int, int);
    Code:
       0: iload_1
       1: iload_2
       2: if_icmple     9
       5: iconst_1
       6: goto          10
       9: iconst_0
      10: ireturn
}
```

Figure 6

Approach

We followed the same approach used by the existing mutators that perform similar replacement such as the Conditional Boundary Mutator. We created a single mutator through an enum that implemented the MethodMutatorFactory which is responsible for creating the MethodVisitor method used to find mutation points and apply the mutators. Each method visitor extends AbstractJumpMutator since it contains the Substitution class which can replace one operator with another.

The flaw of this approach was that each mutator can only produce one mutant per mutation point. At first, we thought to change the way the mutations are stored by changing the map to a MultivaluedMap and the HashMap to a MultivaluedHashMap. However, that would require changing code outside the class we were creating.

We ended up creating a class containing several mutators defined through enums. We also defined method visitor for each of the cases we were going to handle. Below is a snippet of the method visitor responsible for replacing operators with the '!='.

Note that the operators are not equivalent to the signs used. The reason is that JVM flips the signs of the operator with their complement. As you can see in the figure below '>' becomes 'if_icmple'.

Phase 2:

CR Mutator

Constant Replacement Mutator (Figure 8).

Real time example (Figure 9/10)

```
Mutations

1. ROR Mutator: Replaced '<' with '!='. → SURVIVED
2. ROR Mutator: Replaced '<' with '=='. → KILLED
9
3. ROR Mutator: Replaced '<' with '>='. → KILLED
4. ROR Mutator: Replaced '<' with >'. → KILLED
5. ROR Mutator: Replaced '<' with '<='. → KILLED
1. AOD Mutator: Removed the second operator from a subtraction formula (int) → KILLED
2. AOD Mutator: Removed the first operator from a subtraction formula (int) → KILLED
11
3. AOR Mutator: Replaced integer subtraction with addition → KILLED
4. AOR Mutator: Replaced integer subtraction with division → KILLED
5. AOR Mutator: Replaced integer subtraction with multiplication → KILLED
```

Figure 7

Mutation operator	Expression	Mutant
CRMutator.MutantType.ADD	X	X + 1
CRMutator.MutantType.SUB	X	X - 1
CRMutator.MutantType.NEGATE	X	-X
CRMutator.MutantType.REPLACE_ONE	X	1
CRMutator.MutantType.REPLACE_ZERO	X	0

```
155       {
156         // Run in case of any constructor
157         exitValues(DEFAULT_EXIT_VALUES);
158         stopper(DestroyProcessStopper.INSTANCE);
159         redirectOutput(null);
160         redirectError(null);
161         destroyer(null);
162  16     redirectErrorStream(DEFAULT_REDIRECT_ERROR_STREAM);
163       }
164
```

Figure 9

```
162
1. CR Mutator: Incremented value of common constant (visitInsn) → KILLED
2. CR Mutator: Negated value of common constant (visitInsn) → KILLED
3. CR Mutator: Replaced value of common constant with 0 (visitInsn) → KILLED
4. CR Mutator: Decremented value of common constant (visitInsn) → KILLED
5. CR Mutator: Incremented value of common constant (visitInsn) → SURVIVED
6. CR Mutator: Negated value of common constant (visitInsn) → SURVIVED
7. CR Mutator: Replaced value of common constant with 0 (visitInsn) → SURVIVED
8. CR Mutator: Decremented value of common constant (visitInsn) → SURVIVED
9. CR Mutator: Incremented value of common constant (visitInsn) → KILLED
10. CR Mutator: Negated value of common constant (visitInsn) → SURVIVED
11. CR Mutator: Replaced value of common constant with 0 (visitInsn) → KILLED
12. CR Mutator: Decremented value of common constant (visitInsn) → KILLED
13. CR Mutator: Incremented value of common constant (visitInsn) → SURVIVED
14. CR Mutator: Negated value of common constant (visitInsn) → KILLED
15. CR Mutator: Replaced value of common constant with 0 (visitInsn) → KILLED
16. CR Mutator: Decremented value of common constant (visitInsn) → KILLED
```

Figure 10

UOR Mutator:

We implemented the unary operator replacement mutator to target every instance of the increment and decrement operators and replace them with the following:

DECREMENT	INCREMENT	REMOVE	REVERSE

Special cases: (Figure 11)

1. If the 'INCREMENT' operator is target by the 'DECREMENT' operator, it acts as a 'REMOVE'.
2. If the 'INCREMENT' operator is target by the 'INCREMENT' operator, it acts as a 'FURTHER INCREMENT'.
3. The reverse of the two cases listed above is true for the 'DECREMENT' operator.

Approach:
The implementation of this mutator was based on the already existing NegateVariableMutator which essentially performs the 'REVERSE' mutation. It makes use of the visitIincInsn which takes the index of the local variable to be incremented and the amount to increment the local variable by as arguments.

In order to track the sign of the variable to be mutated, we check whether or not the increment is positive or negative. We also refined the way we create the mutator. Rather than creating one enum per mutator, we created a class that implements the MethodMutatorFactory and then defined the enums within the class.

```java
@Override
public void visitIincInsn(final int var, final int increment) {
    String operator;
    String operator2;
    if (increment > 0) {
        operator = "++";
        operator2 = "--";
    } else {
        operator = "--";
        operator2 = "++";
    }

    final MutationIdentifier newId = this.context.registerMutation(this.factory,
            "UOR Mutator: Changed operator from " + operator + " to " + operator2);
    if (this.context.shouldMutate(newId)) {
        this.mv.visitIincInsn(var, -increment);
    } else {
        this.mv.visitIincInsn(var, increment);
    }
}
```

Figure 11

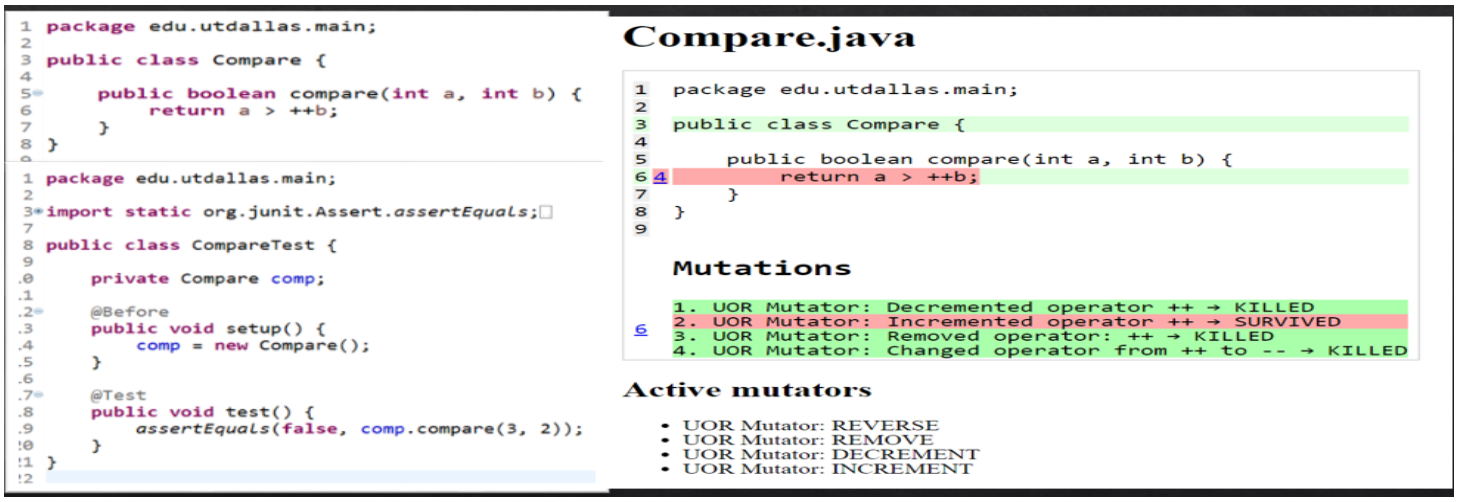


Figure 12

Project	Number of Class	Line Coverage		Mutation Coverage		Running Time	Number of Threads
		percentage	coverage line	percentage	killed/numberOfMutation		
GoogleAuth,7051a9a23913c9ef3808467b1c647cab7ada82cf, https://github.com/wstrange/GoogleAuth	4	73%	187/256	50%	286/576	02:54 min	4

Figure 13

Final part: Mutation can fix bug!

Goal: We aim to use mutation with regard to fixing buggy!

1. Setting

Project: We chose a real small project to run our implementation. The project details are listed :

Name: ***GoogleAuth***

Commit: ***7051a9a23913c9ef3808467b1c647cab7ada82cf***

Github link: ***https://github.com/wstrange/GoogleAuth***

Running mutation testing using PIT on this project, we got: (Figure 13)

2. Bug generation

Because the chosen project is working well, we decided to make a bug in this project.

The bug is manually generated at function : *calculateCode* of the class *GoogleAuthenticator* in this project as below:

10.

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
4	73%	187/256	50%	286/576

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
com.warrenstrange.googleauth	4	73%	187/256	50%	286/576

Report generated by PIT 1.4.0-SNAPSHOT

Figure 14

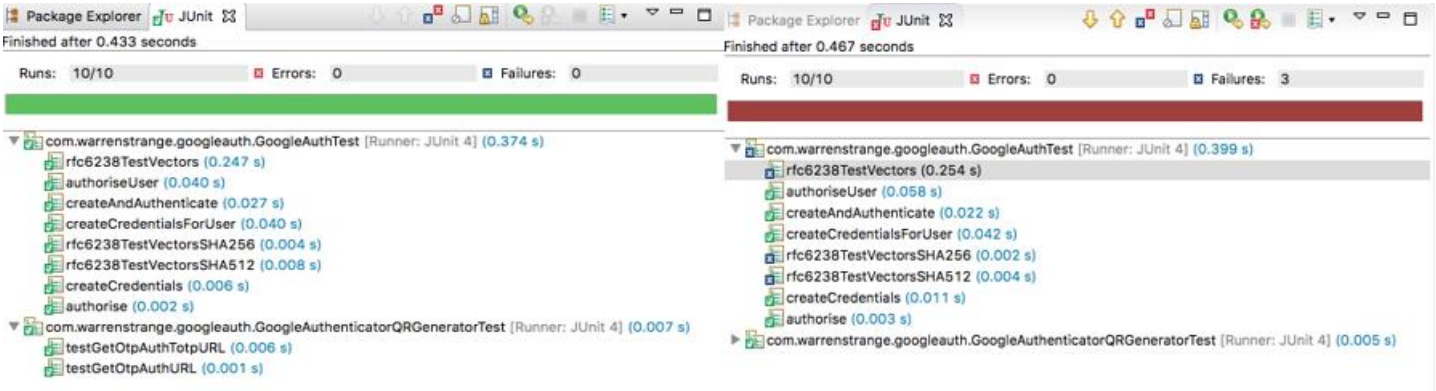


Figure 15

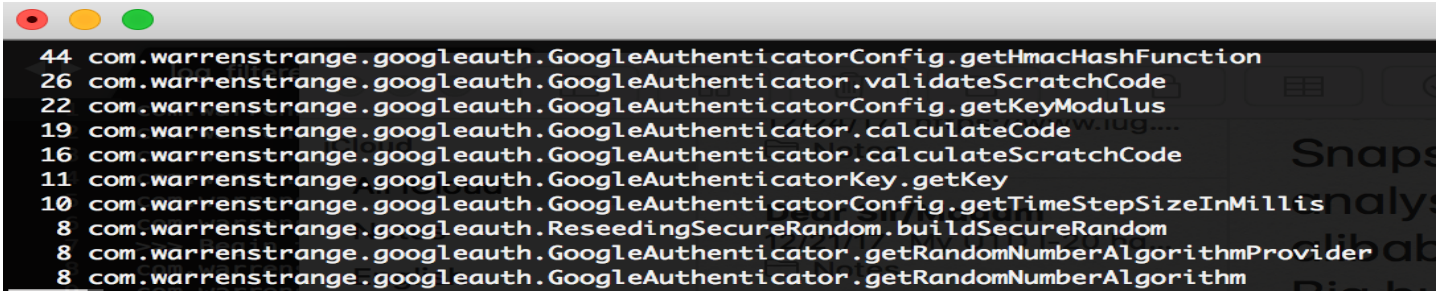
```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <configuration>
     <argLine>
     -javaagent:/Users/tranhieu/Desktop/course-project/method-coverage/target/method-coverage-0.1-SNAPSHOT.jar
     </argLine>
    </configuration>
</plugin>
```

Figure 16

11. data[i] = (**byte**) value; } ...

Original version:

```
1.  int calculateCode(byte[] key, long tm) {
2.
    // Allocating an array of bytes to represent the specified insta
    nt // of time.
3.      byte[] data = new byte[8];
4.
5.      long value = tm;
6.
7.
    // Converting the instant of time from the long representation
     to a
8.      // big-endian array of bytes (RFC4226, 5.2. Description).
9.      for (int i = 8; i-- > 0; value >>>= 8) {
```



Buggy version:

```
1.  int calculateCode(byte[] key, long tm) {
2.
    // Allocating an array of bytes to represent the specified insta
    nt // of time.
3.      byte[] data = new byte[8];
4.
5.      long value = tm + 1; // THIS IS BUGGY
```

Figure 17

```
6.
7.
    // Converting the instant of time from the long representation
     to a
8.      // big-endian array of bytes (RFC4226, 5.2. Description).
9.      for (int i = 8; i-- > 0; value >>>= 8) {
10.       data[i] = (byte) value;
11.     }
12.     ...
```

Running Junit on the original version: (Figure 15)

Running Junit on the buggy version: (Figure 16)

3. Fault localization

Firstly, we need to locate what functions ***might*** cause buggy. To do that, we implement a program using Java Agent ASM to visit class and track in which classes our program run into when we run a certain test. This program called Method-Coverage. Then we build this program with *mvn install*, and add its SNAPSHOT into pom file of the buggy project. Specifically the project here is ***GoogleAuth***

Then, we run **mvn test** in the directory of the buggy project. Using Method-Coverage we can get a log file like that: (Figure 17 and Figure 18)

This log file shows us that tests. We then filter out extra line to get only useful information from log using command:

```
1.  tranhieu@ HieuTran - MacBook - Pro: ~/Desktop%code%
    nbsp;cat logTest | grep -v -E "^org" | grep -v -
    E "^javax" | grep -v -E "^false" | sed '/ ^  % code % nbsp;
2.  / d ' | fgrep -v -E "^\[" | grep -v -E "^junit" | grep -
    E "^com\.|^>>>" > log_filtered
```

```
2069    javax.crypto.JceSecurity.canUseProvider
2070    javax.crypto.JceSecurity.getVerificationResult
2071    javax/crypto/MacSpi
2072    false
2073    javax.crypto.MacSpi.<init>
2074    javax.crypto.spec.SecretKeySpec.getEncoded
2075    javax.crypto.Mac.doFinal
2076    javax.crypto.Mac.chooseFirstProvider
2077    javax.crypto.Mac.update
2078    javax.crypto.Mac.chooseFirstProvider
2079    javax.crypto.Mac.doFinal
2080    javax.crypto.Mac.chooseFirstProvider
2081    com.warrenstrange.googleauth.GoogleAuthenticatorConfig.getKeyModulus
2082    >>> ERROR testing0: rfc6238TestVectors
2083    org.junit.Assert.assertEquals
2084    org.junit.Assert.assertEquals
```

Figure 18

```
253     com.warrenstrange.googleauth.GoogleAuthenticatorConfig.getHmacHashFunction
254     com.warrenstrange.googleauth.GoogleAuthenticatorConfig.getKeyModulus
255     com.warrenstrange.googleauth.GoogleAuthenticatorQRGeneratorTest.setUp
256     com.warrenstrange.googleauth.GoogleAuthenticatorQRGeneratorTest.testGetOtpAuthTotpURL
257     >>> Begin testing1: testGetOtpAuthTotpURL
258     com.warrenstrange.googleauth.GoogleAuthenticatorQRGenerator.getOtpAuthTotpURL
259     com.warrenstrange.googleauth.GoogleAuthenticatorQRGenerator.formatLabel
260     com.warrenstrange.googleauth.GoogleAuthenticatorKey.getKey
261     com.warrenstrange.googleauth.GoogleAuthenticatorKey.getConfig
262     com.warrenstrange.googleauth.GoogleAuthenticatorConfig.getHmacHashFunction
263     com.warrenstrange.googleauth.GoogleAuthenticatorQRGenerator.getAlgorithmName
264     com.warrenstrange.googleauth.GoogleAuthenticatorConfig.getCodeDigits
265     com.warrenstrange.googleauth.GoogleAuthenticatorConfig.getTimeStepSizeInMillis
266     >>> Finish testing1: testGetOtpAuthTotpURL
267     >>> Begin testing2: testGetOtpAuthTotpURL
268     com.warrenstrange.googleauth.GoogleAuthenticatorQRGenerator.getOtpAuthTotpURL
269     com.warrenstrange.googleauth.GoogleAuthenticatorQRGenerator.formatLabel
270     com.warrenstrange.googleauth.GoogleAuthenticatorKey.getKey
271     com.warrenstrange.googleauth.GoogleAuthenticatorKey.getConfig
272     com.warrenstrange.googleauth.GoogleAuthenticatorConfig.getHmacHashFunction
273     com.warrenstrange.googleauth.GoogleAuthenticatorQRGenerator.getAlgorithmName
274     com.warrenstrange.googleauth.GoogleAuthenticatorConfig.getCodeDigits
275     com.warrenstrange.googleauth.GoogleAuthenticatorConfig.getTimeStepSizeInMillis
276     >>> Finish testing2: testGetOtpAuthTotpURL
277     >>> Begin testing3: testGetOtpAuthTotpURL
278     com.warrenstrange.googleauth.GoogleAuthenticatorQRGenerator.getOtpAuthTotpURL
279     com.warrenstrange.googleauth.GoogleAuthenticatorQRGenerator.formatLabel
280     com.warrenstrange.googleauth.GoogleAuthenticatorKey.getKey
281     com.warrenstrange.googleauth.GoogleAuthenticatorKey.getConfig
282     com.warrenstrange.googleauth.GoogleAuthenticatorConfig.getHmacHashFunction
283     com.warrenstrange.googleauth.GoogleAuthenticatorQRGenerator.getAlgorithmName
284     com.warrenstrange.googleauth.GoogleAuthenticatorConfig.getCodeDigits
285     com.warrenstrange.googleauth.GoogleAuthenticatorConfig.getTimeStepSizeInMillis
286     >>> Finish testing3: testGetOtpAuthTotpURL
287     com.warrenstrange.googleauth.GoogleAuthenticatorQRGeneratorTest.setUp
288     com.warrenstrange.googleauth.GoogleAuthenticatorQRGeneratorTest.testGetOtpAuthURL
289     >>> Begining testing: testGetOtpAuthURL
290     com.warrenstrange.googleauth.GoogleAuthenticatorQRGenerator.getOtpAuthURL
```

Figure 19

```
5093    org.junit.runner.Description.getDisplayName
5094    org.junit.runner.notification.Failure.getDescription
5095    org.junit.runner.Description.getDisplayName
5096    org.junit.runner.notification.Failure.getDescription
5097    org.junit.runner.Description.getDisplayName
5098    org.junit.runner.notification.RunNotifier.<init>
5099    org.junit.runner.notification.RunNotifier.removeListener
5100    org.junit.runner.notification.RunNotifier.wrapIfNotThreadSafe
5101    org.junit.runner.notification.RunListener.<init>
5102
5103    Results :
5104
5105    Failed tests:   rfc6238TestVectors(com.warrenstrange.googleauth.GoogleAuthTest): expected:<37359152> but was:<94287082>
5106      rfc6238TestVectorsSHA256(com.warrenstrange.googleauth.GoogleAuthTest): expected:<30882438> but was:<46119246>
5107      rfc6238TestVectorsSHA512(com.warrenstrange.googleauth.GoogleAuthTest): expected:<68765371> but was:<90693936>
5108
5109    Tests run: 10, Failures: 3, Errors: 0, Skipped: 0
5110
5111    [INFO] ------------------------------------------------------------------------
5112    [INFO] BUILD FAILURE
5113    [INFO] ------------------------------------------------------------------------
5114    [INFO] Total time: 2.767 s
5115    [INFO] Finished at: 2018-04-28T19:34:27-05:00
5116    [INFO] ------------------------------------------------------------------------
5117    [ERROR] Failed to execute goal org.apache.maven.plugins:maven-surefire-plugin:2.12.4:test (default-test) on project googleauth: There are test failures.
5118    [ERROR]
5119    [ERROR] Please refer to /Users/tranhieu/Desktop/project/testing_project/GoogleAuth-master/target/surefire-reports for the individual test results.
5120    [ERROR] -> [Help 1]
5121    [ERROR]
5122    [ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
5123    [ERROR] Re-run Maven using the -X switch to enable full debug logging.
5124    [ERROR]
5125    [ERROR] For more information about the errors and possible solutions, please read the following articles:
5126    [ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException
5127
```

Figure 20

After filtering method call of library, we can have a list of methods called when running tests:
Example: (Figure 19)

Top 10 methods called during running test suit (the first column is the number of appearance: (Figure 20)

Using fault localization for these method, we ranked this methods by formula Tarantula:

$$Suspicious\ (s) = \frac{fail\ (s)/totalfail}{fail\ (s)/totalfail\ +\ pass\ (s)/totalpass}$$

Where fails – the number of failed test on this method
totalfail – the number of failed test on the project (= 3 for this project)
Pass – the number of passed test on this method
totalpass – the number of passed test on the project (= 10 for this project)
The table below is top-3 suspicious methods:

3. Mutation for fixing bug

Now we run mutation on top-3 these methods.
Because we made up the buggy :
long value = tm; => **long** value = tm + 1;
So we use mutation AOD that replace the second part of expression by first/second operators on these methods.
Each time, we made only 1 AOD mutant for a method. Then we re-run the Junit test. If all the tests are passed, that means that the mutant has fixed the bug.
In our case, after the first AOD mutant of the class *calculateCode,* the second part of the expression has been replaced by its first operator, we got a mutant that makes the project passing all the test. So that the mutant has fixed the buggy.

Method	Suspicious score
com.warrenstrange.googleauth.GoogleAuthenticator.calculateCode	0.833
com.warrenstrange.googleauth.GoogleAuthenticatorConfig.getKeyModulus	0.71
com.warrenstrange.googleauth.GoogleAuthenticatorConfig.getHmacHashFunction	0.5

```
210         * @param key the secret key in binary format.
211         * @param tm  the instant of time.
212         * @return the validation code for the provided key at the specified instant
213         * of time.
214         */
215        int calculateCode(byte[] key, long tm)
216        {
217            // Allocating an array of bytes to represent the specified instant
218            // of time.
219            byte[] data = new byte[8];
220 1          long value = tm + 1;
221
222            // Converting the instant of time from the long representation to a
```

Figure 21

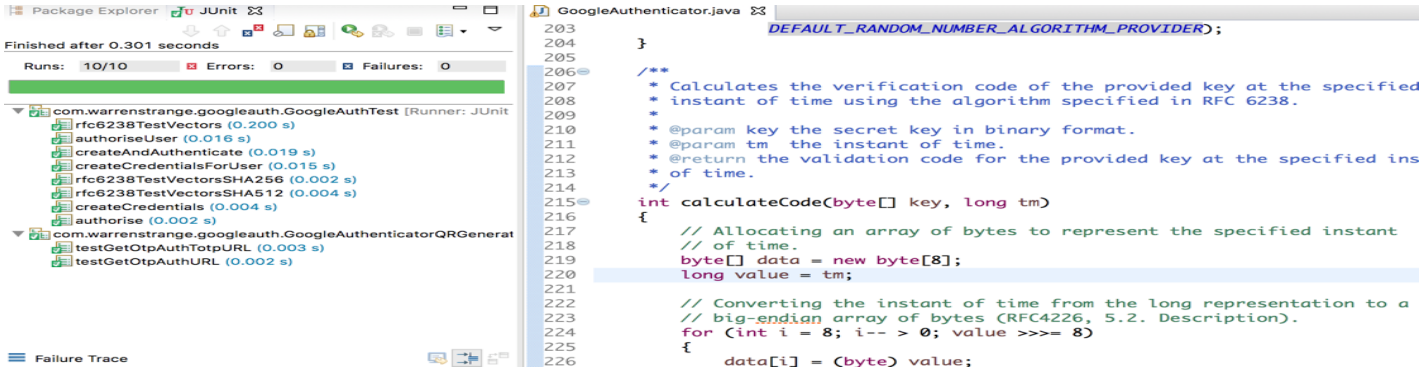


Figure 22

Project	Number of Class	Line Coverage		Mutation Coverage		Running Time	Number of Threads
		percentage	coverage line	percentage	killed/numberOfMutation		
zt-exec,305172eaed27aa71a6f4de970d20c73cefe6291e,https://github.com/zerotu rnaround/zt-exec	13	59%	404/689	29%	202/699	18:16 min	4
aho-corasick,25eeef5168846d50dc343c1f224a24745f925f5b,https://github.com/ro bert-bor/aho-corasick	12	94%	322/341	70%	481/686	0:46 min	4
GoogleAuth,7051a9a23913c9ef3808467b1c647cab7ada82cf,https://github.co m/wstrange/GoogleAuth	4	73%	187/256	50%	286/576	02:54 min	4
RestFixture,bb4c7071c4edfd9f4aac8c6dd3aee1b196ffa0f1,https://github.com/ smartrics/RestFixture	28	78%	1290/1655	53%	806/1513	02:08 min	4
pollexor,0255b9bbd0163dc2e3c86b3bfc9127ecf3a01cfc,https://github.com/sq uare/pollexor	3	90%	262/292	80%	756/945	0:38 min	4
rtree,e406cff766740b117898d6777c094ed494220fa5,https://github.com/david moten/rtree	71	80%	1868/2349	66%	1243/1874	36:16 min	4
Project given by TA							
commons-codec	51	92%	3592/3906	78%	14955/19122	18:24 min	4
commons-lang	106	95%	13894/14563	73%	35563/48766	02:23 h	4
jfreechart	Got crash after running more than 2 hours						4
joda-time							4

In summary, our strategy for fixing buggy:
Tracking methods called for every tests.
Using fault localization and ranking by Tarantula to rank top suspicious methods.
Generating only 1 mutant each time on one of the top-suspicious methods
Re-run the test suit, if all test are passed, then terminating, else go back to step 3.

Results
We first tested our mutators on the program we created.

Future plans
For the second phase, we are planning to implement the four mutators listed on the project guidelines and possibly some additional ones as time permits

Results of Tested projects:
The following table represents the Testing results of three random projects from github on our mutation tester program.

Github Reporsitory to project:
https://github.com/Oxiras/PIT

References:

[1] E. W. Dijkstra, Notes On Structured Programming, Section 3 ("On The Reliability of Mechanisms"), 1970.
[2] http://pitest.org/