**Final part: Mutation can fix bug!**

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

1. **Setting**

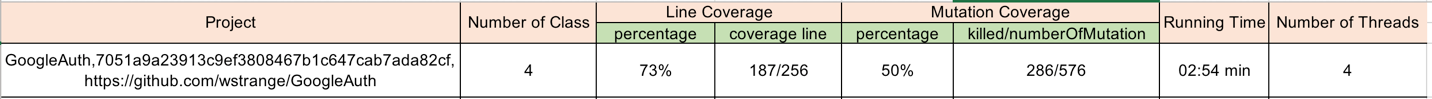
**Project**: We choose a real small project to run our implementation. The project is

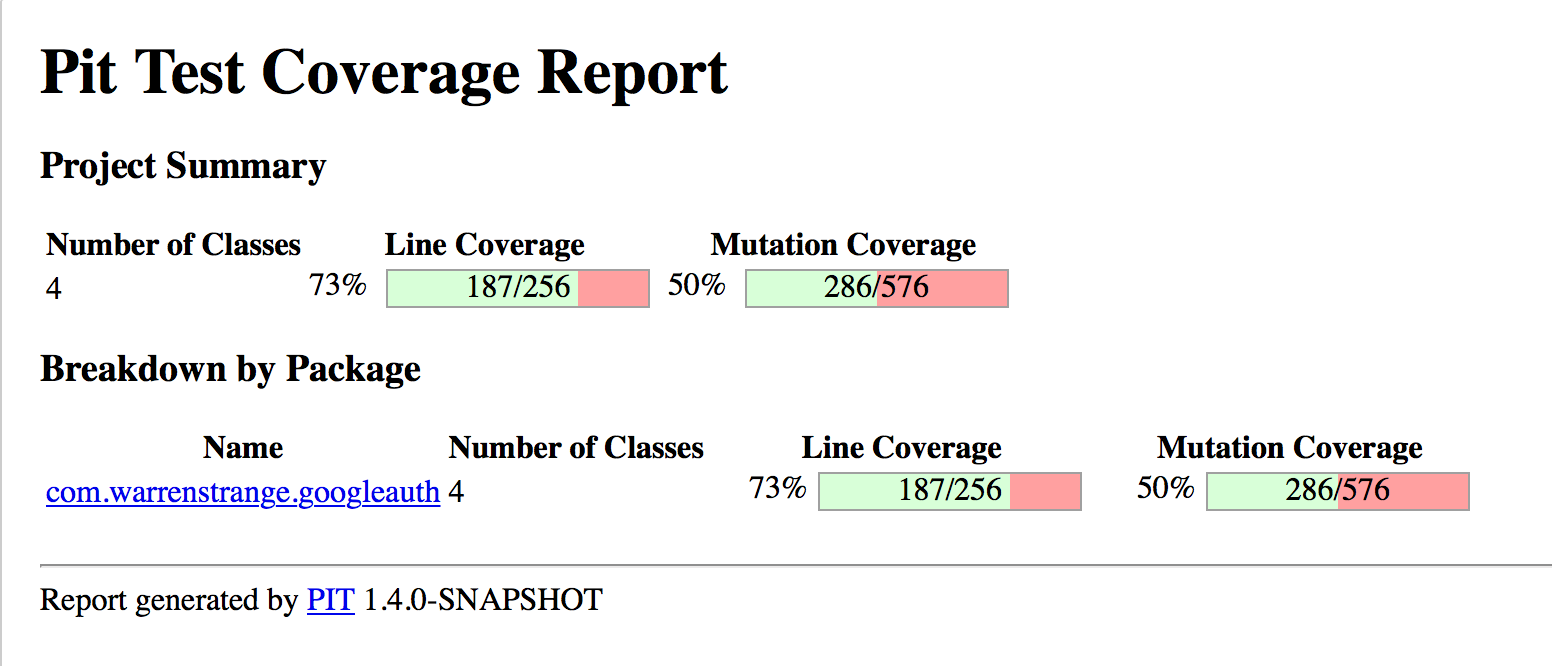
Name: ***GoogleAuth***

Commit: ***7051a9a23913c9ef3808467b1c647cab7ada82cf***

Github link: [***https://github.com/wstrange/GoogleAuth***](https://github.com/wstrange/GoogleAuth)

Running mutation testing using PIT on this project, we got:





1. **Bug generation**

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

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

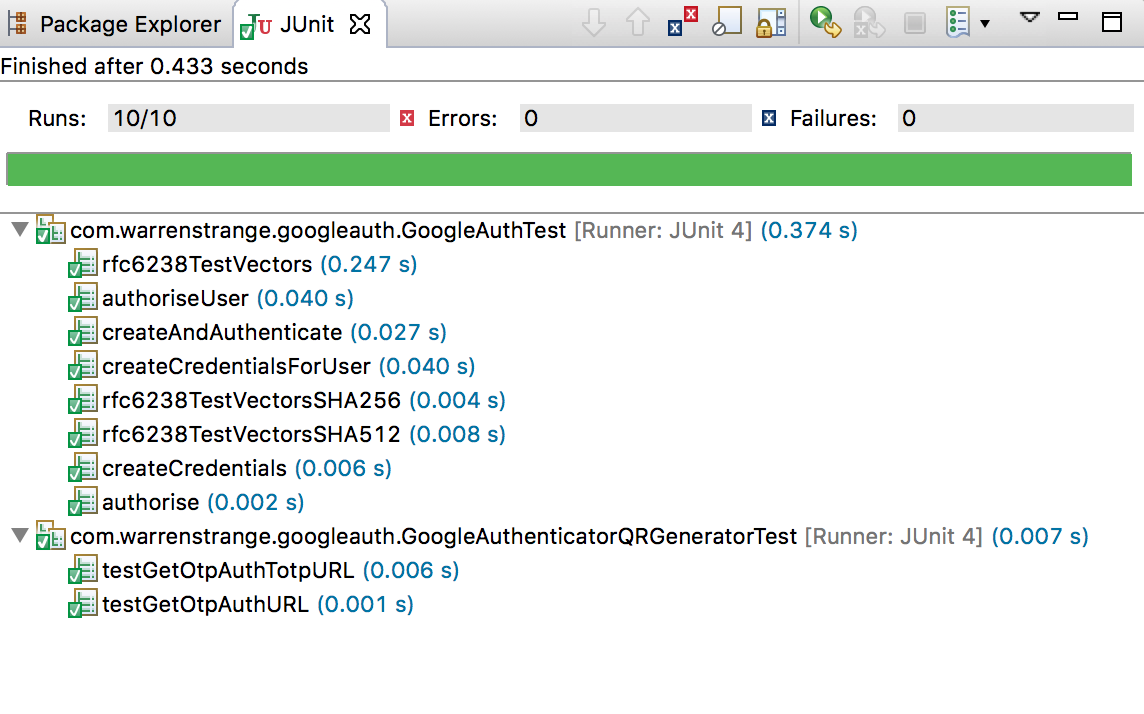
Original version:

1. int calculateCode(byte[] key, long tm) {
2. // Allocating an array of bytes to represent the specified instant // of time.
3. byte[] data = new byte[8];
4. long value = tm;
5. // Converting the instant of time from the long representation to a
6. // big-endian array of bytes (RFC4226, 5.2. Description).
7. for (int i = 8; i-- > 0; value >>>= 8) {
8. data[i] = (byte) value;
9. }
10. …

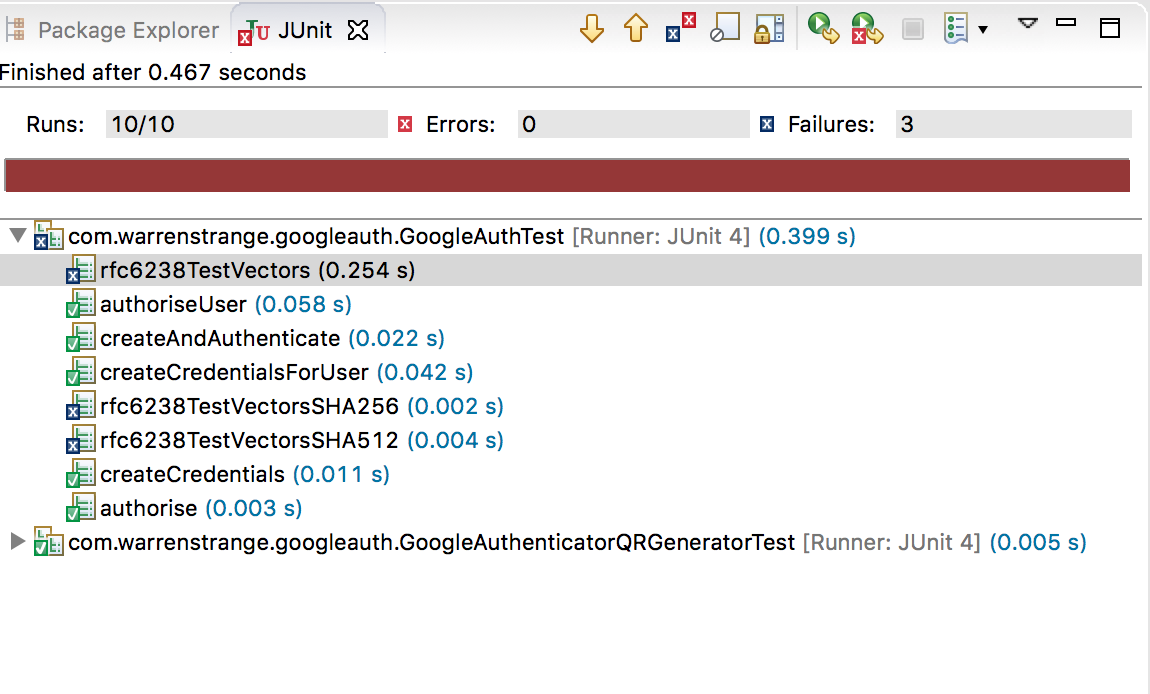
Buggy version:

1. int calculateCode(byte[] key, long tm) {
2. // Allocating an array of bytes to represent the specified instant // of time.
3. byte[] data = new byte[8];
4. long value = tm + 1; // THIS IS BUGGY
5. // Converting the instant of time from the long representation to a
6. // big-endian array of bytes (RFC4226, 5.2. Description).
7. for (int i = 8; i-- > 0; value >>>= 8) {
8. data[i] = (byte) value;
9. }
10. …

Running Junit on the original version:



Running Junit on the buggy version:

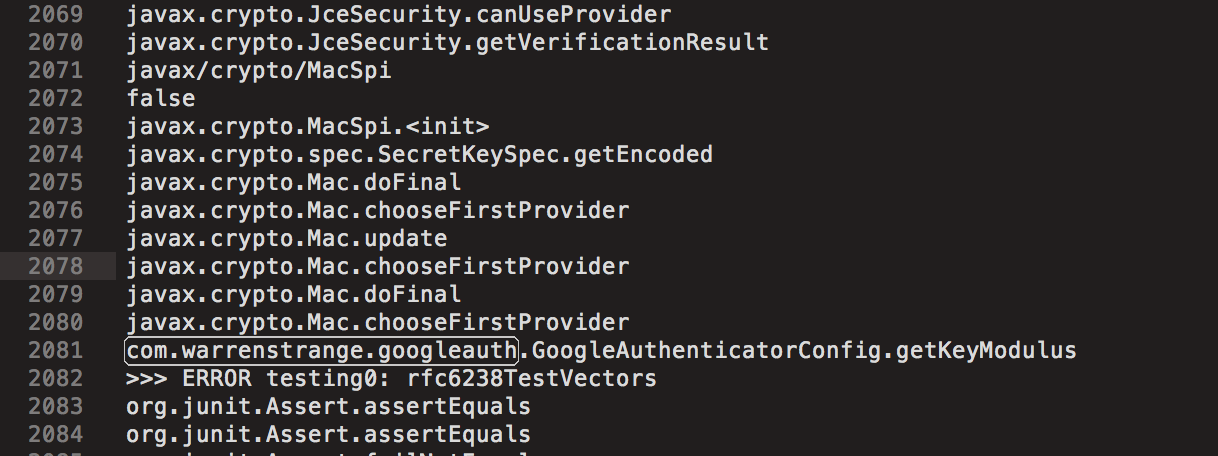


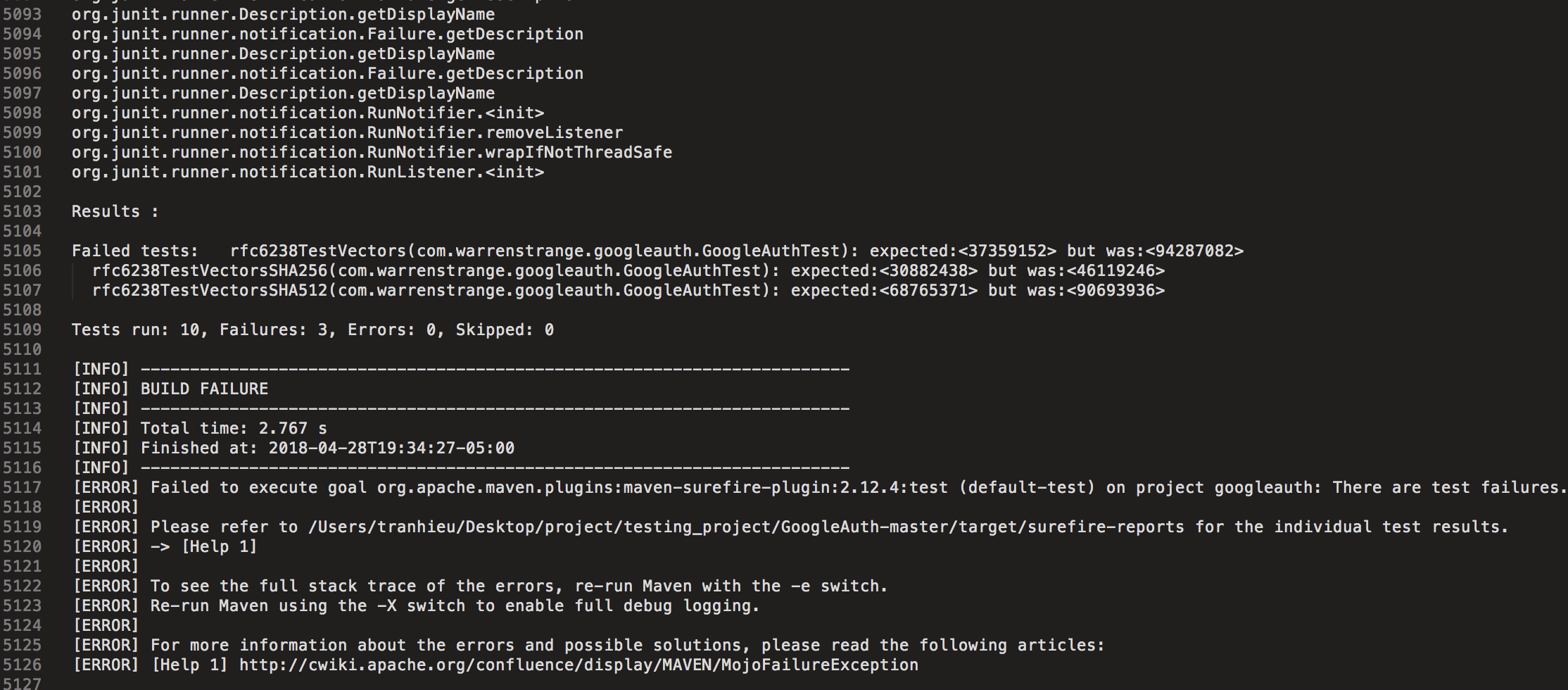
1. **Fault localization**

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:



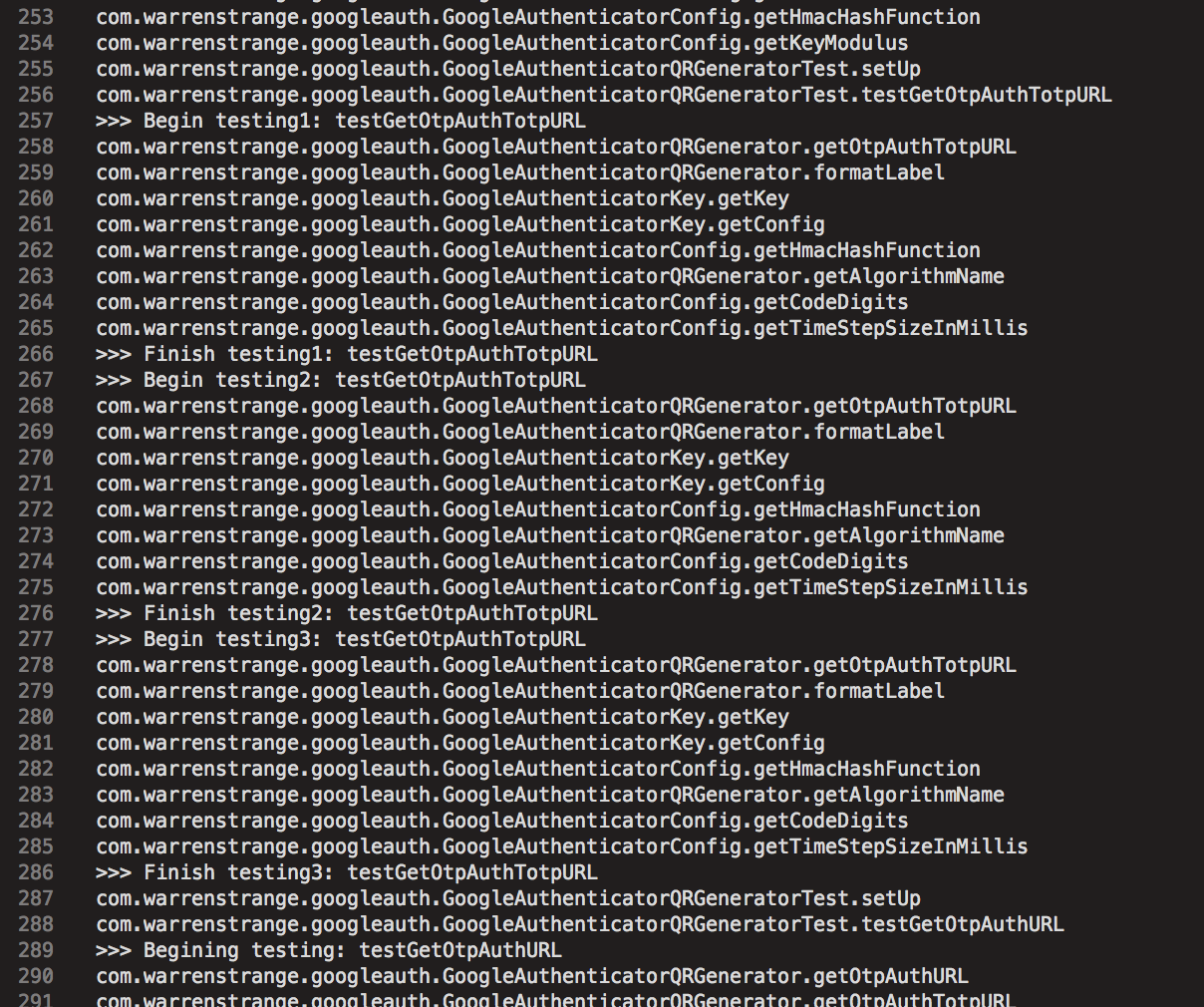


This log file shows us all functions are executed when running that tests. We then filter out extra line to get only useful information from log using command:

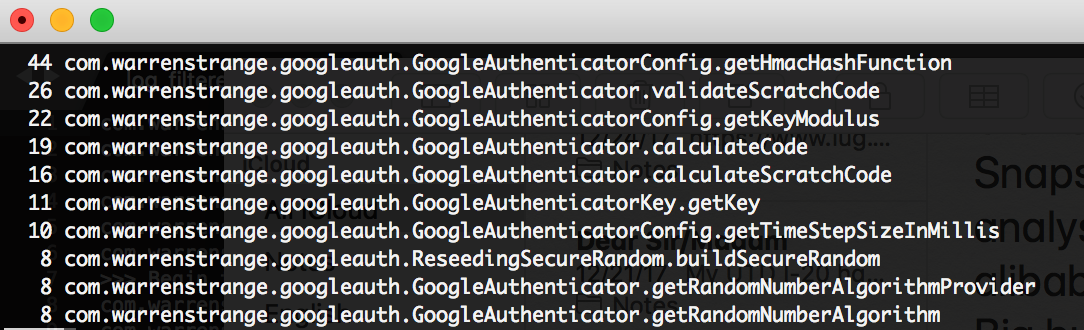
1. tranhieu@  HieuTran  -  MacBook  -  Pro:  ~/Desktop%code%nbsp;cat logTest | grep -v -E "^org" | grep -v -E "^javax" | grep -v -E "^false" | sed '/  ^   % code % nbsp;
2. / d ' | fgrep -v -E "^\[" | grep -v -E "^junit" | grep -E "^com\.|^>>>" > log\_filtered

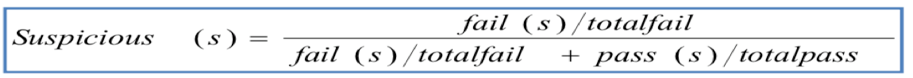
After filtering method call of library, we can have a list of methods called when running tests:

Example:



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



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

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:

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

1. **Mutation for fixing bug**

Now we run mutation on top-3 these methods.

Because we made up the buggy :

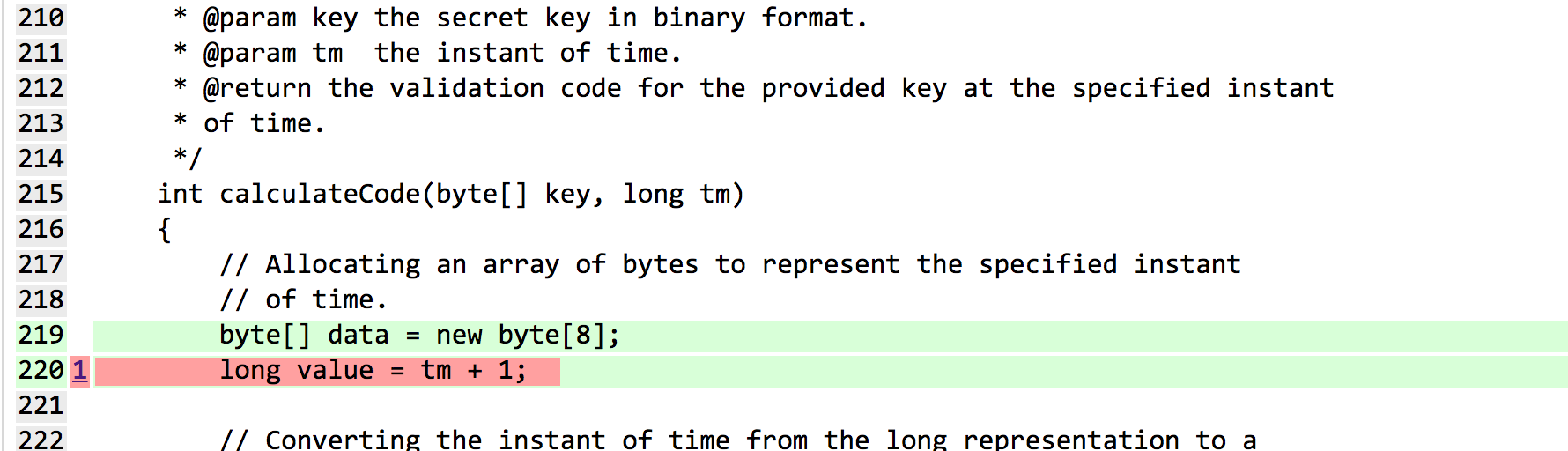
long value = tm; => long value = tm + 1;

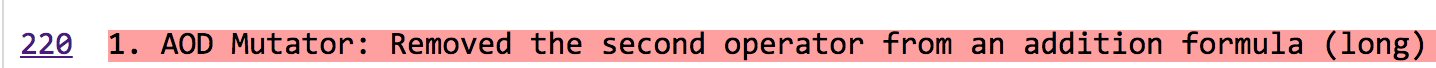
So we use mutation AOD that replace the second part of expression by first/second operators on these methods.

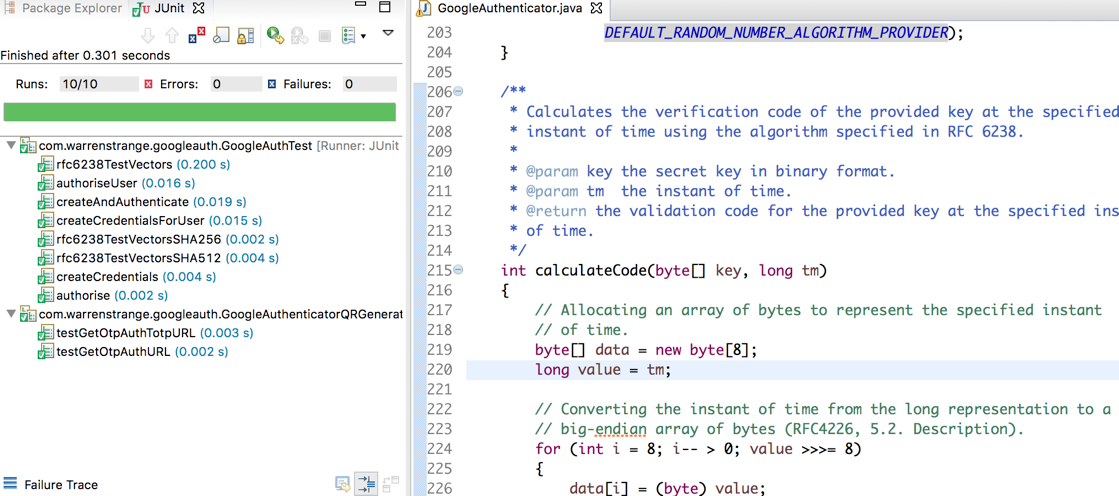
Each time, we made only 1 AOD mutant for a method. Then we re-run the Junit test. If all the tests are passed, that means that the mutant has fixed the bug.

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.

Mutant:







**In summary, our strategy for fixing buggy:**

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