# CSE364 Software Engineering Individual Assignment

Truong Huy Trung[1]

[1]Department of Computer Science and Engineering, UNIST
truonghuytrung@unist.ac.kr

May 22, 2023

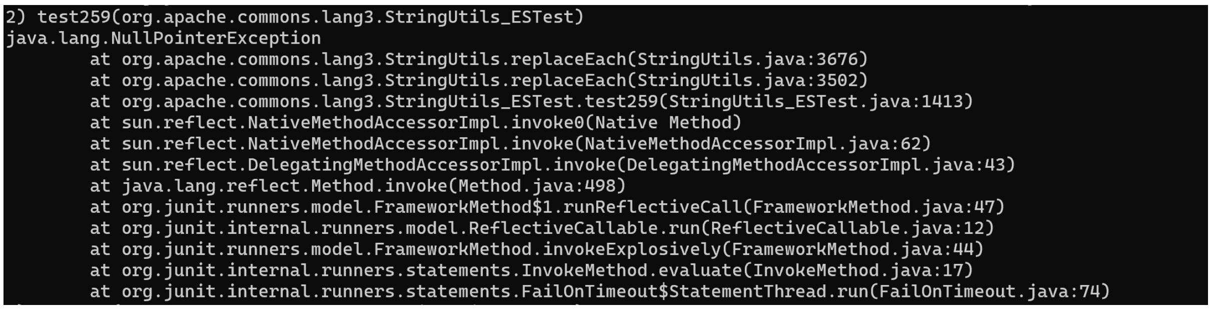
## 1 Detect the localize a bug

While running the Evosuite and Randoop-generated tests. I figured that the amount of failed tests produced by Randoop was so much larger than Evosuite in terms of the number of failed tests. Therefore, I chose to focus on the tests generated by Evosuite.

**Randoop**: Tests run: 1466, Failures: 1466.

**Evosuite**: Tests run: 412, Failures: 8

Taking a closer look at the failed tests in Evosuite, one particular test caught my eye because of its distinction from the other error messages:



```
2) test259(org.apache.commons.lang3.StringUtils_ESTest)
java.lang.NullPointerException
        at org.apache.commons.lang3.StringUtils.replaceEach(StringUtils.java:3676)
        at org.apache.commons.lang3.StringUtils.replaceEach(StringUtils.java:3502)
        at org.apache.commons.lang3.StringUtils_ESTest.test259(StringUtils_ESTest.java:1413)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:47)
        at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
        at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:44)
        at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
        at org.junit.internal.runners.statements.FailOnTimeout$StatementThread.run(FailOnTimeout.java:74)
```

Figure 1: The suspected test

This failed test is related to the **replaceEach** method in the StringUtils.java, and after tracing the logic of the method with the input, I discovered that the bug is from this code snippet:

```
private static String replaceEach(String text, String[] searchList, String[]
                              replacementList, boolean repeat, int timeToLive)
{
                                      ...
    if (searchLength != replacementLength) {
        throw new IllegalArgumentException("Search and Replace array lengths don't match: "
            + searchLength
            + " vs "
            + replacementLength);
    }
                                      ...
    //the actual for-loop that caused of the bug
    for (int i = 0; i < searchList.length; i++) {
        int greater = replacementList[i].length() - searchList[i].length();
        if (greater > 0) {
            increase += 3 * greater; // assume 3 matches
        }
    }
                                        ...
}
```

## 1.1 How this bug occurred and its root cause

The parent method assumes the **searchList** and the **replacementList** have the same length. Therefore, in this snippet, this is equivalent to accessing each element of **replacementList** and performing some logic on it. However, it does not cover the case where there can be some element of the **replacementList** that is null. Which led to the error **java.lang.NullPointerException**.

Now taking a look at the test:

```
@Test(timeout = 4000)
public void test259() throws Throwable {
    String[] stringArray0 = new String[3];
    stringArray0[0] = "WmB:(";
    stringArray0[1] = "WmB:(";
    // Undeclared exception!
    StringUtils.replaceEach("WmB:(", stringArray0, stringArray0);
}
```

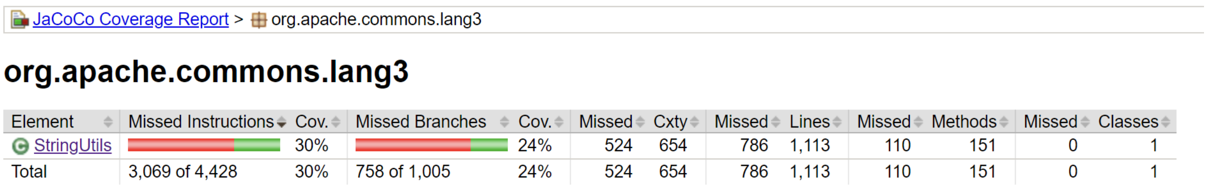
We can see that **stringArray0**[**2**] is null, but this is still a valid input because it has been specified **@param replacementList: the Strings to replace them with, no-op if null**. No-op has not been chosen, instead, it raised an error. Therefore, I was able to confirm that this is a bug.

## 1.2 Which tool did you use to reveal the bug? Did the other one succeed to reveal the same bug as well?

I used Evosuite to reveal this bug. About the tests generated by Randoop, after 20+ times of re-generating with different time limits. I was still not able to create a test that can reproduce the issue.

# 2 Measure coverage of generated tests

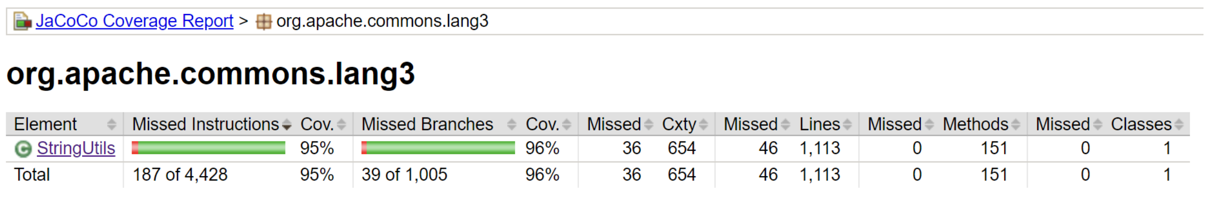
These are the coverage of 2 used tools:


JaCoCo Coverage Report > org.apache.commons.lang3

org.apache.commons.lang3

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| StringUtils | 🟥🟩 | 30% | 🟥🟩 | 24% | 524 | 654 | 786 | 1,113 | 110 | 151 | 0 | 1 |
| Total | 3,069 of 4,428 | 30% | 758 of 1,005 | 24% | 524 | 654 | 786 | 1,113 | 110 | 151 | 0 | 1 |

Figure 2: Randoop Coverage Report



JaCoCo Coverage Report > org.apache.commons.lang3

org.apache.commons.lang3

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| StringUtils | 🟥🟩 | 95% | 🟥🟩 | 96% | 36 | 654 | 46 | 1,113 | 0 | 151 | 0 | 1 |
| Total | 187 of 4,428 | 95% | 39 of 1,005 | 96% | 36 | 654 | 46 | 1,113 | 0 | 151 | 0 | 1 |

Figure 3: Evosuite Coverage Report

## 3 Describe strengths and weaknesses of Randoop and Evosuite

### 3.1 Randoop

For Randoop, there are a few strengths:

- Randoop works on the basis of feedback directed testing. This allows Randoop to create tests guided by the feedback obtained from the tests created before.
- Randoop can decide whether the newly generated tests are redundant/illegal or not and then discard them if they are.
- Randoop's output only test cases which are failed by the program.
- Easy to use.

The weakness of Randoop are:

- High cost for test code maintenance. Although the test cases are generated pretty quickly, it contains a lot of lines of code in total.
- Low coverage. As attached in the previous section, Randoop has a low coverage score of 24% despite a large number of tests. However, this can be increased if the time limit increases.
- High amount of false alarms tests.

### 3.2 Evosuite

For Evosuite, there are a few strengths:

- Can easily obtain high code coverage because test generation is guided by coverage.
- Evosuite can handle complex scenarios involving external dependencies because it can utilize mocking frameworks.
- Can be integrated with popular IDEs such as Eclipse or IntelliJ
- Constantly evolve its test suite to better fit the end goal.

The weakness of Evosuite are:

- Likely to produce a lot of false alarms by generating invalid inputs for test cases.
- Slower than Randoop, taking 1.38 times longer with the command specified in the handout.

## 4 Improve Test Generation

### 4.1 Randoop

For Randoop, there is something that can be improved:

The feedback mechanism now only consists of 3 steps: Is the test valid, is it redundant, and does it violate the contract? I think it can provide another step to check if the method input for the test is valid.

To do this Randoop can integrate some basic constraints for every input, for example, for each array declaration, indexes used as input can only be in a certain range, etc...

### 4.2 Evosuite

For Evosuite, there is something that can be improved:

Since Evosuite tends to generate false alarms, we can tackle this issue by adding functionality where test cases that are similar to a false alarm are discarded during the test generation process.

More specifically, when a test suite is generated, users can identify manually the false alarms, and re-run Evosuite with a flag that excludes these false alarms. Therefore, Evosuite's new run will simply discard tests that are similar to the false alarms during its process, which results in a better test suite eventually. A similarity score can be computed between the flagged tests and the generated tests to know which ones are to be discarded.

About how to exclude these false alarm tests during the process, there needs to be extra implementation for the tool to utilize this functionality. A choice can be utilizing static syntax analysis to compare the generated tests to the flagged false alarms.

## 5 References

- G. Fraser and A. Arcuri, "Evolutionary Generation of Whole Test Suites," in International Conference On Quality Software (QSIC), Los Alamitos, CA, USA, 2011, pp. 31-40.
- Pacheco, C., Ernst, M.D.: Randoop: feedback-directed random testing for java. In: Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion. pp. 815–816. ACM (2007)