CS4218 Software Testing Report - Milestone 2

1. Test Driven Development

Before introducing our TDD process, we need to make a claim that we could not pass all original given test cases, since the original tdd test itself is buggy. After solving the bugs and making some improvements, it indeed helps us to do TDD.

1.1. Details

While doing TDD, we also keep using github issues and the project panel for collaborating. When a failed test reveals a possible bug, we create an issue for it and assign the issue to our team member. We first used the official test suite to search for bugs in our implementation, after that, we created integration test cases to further reveal the bugs. We successfully found some bugs and assumptions that we didn't think about, and improved our implementation accordingly.

1.2. Bug revealed

We found bugs using both unit testing and integration testing, most bugs are from the EF1 applications which we didn't implement in milestone 1, or the shell level functions that duel with the interaction between different commands.

2. Integration Testing

Before introducing our Integration Testing details, as we didn't learn too much about integration testing in practice before Milestone 1, we wrongly used system testing like test cases as the integration testing. And now, we rename the old "IntegrationTest" to "SystemTest". And we created another 34 integration tests.

2.1. Plan

Our integration testing plan are mainly three parts:

1. Use all ready to use integration test cases on the project description document.
2. Try pair combinations of applications and command operators.
    a. pairwise among Command operators
    b. pairwise among BF Applications
    c. pairwise among EF1 Applications
    d. pairwise among EF2 Applications

   e. pairwise among all Applications
3. Along with integration testing, we try to add both positive and negative tests.

To show the pairwise testing plan in detail, some tests are shown in the tables provided below.

(PS: the status is the pass/fail result of the first time run)

2.2. Execution

a. Pairwise Testing among non- Applications

Status	Operator1	Operator2	Application(s)	Positive/ Negative	Info
√	pipe: |	globbing: *	ls, grep	positive	
	ls src/test/IntegrationTest/testFiles/testFiles/test* | grep '1'				
√	pipe: |	IO: >	paste, wc	positive	
	paste src/test/IntegrationTest/testFiles/test1.txt | wc > src/test/IntegrationTest/testFiles/result.txt				
√	pipe: |	substitution: ``	paste, grep	positive	
	paste src/test/IntegrationTest/testFiles/test1.txt | grep `echo 'wor'`				
x	seq: ;	globbing: *	cd, ls	positive	add assumption for ls
	cd src/test/IntegrationTest/testFiles; ls test*				
x	seq: ;	substitution: ``	wc	positive	add assumption for wc
	wc src/test/IntegrationTest/testFiles/test* > src/test/IntegrationTest/testFiles/result.txt				

b. Pairwise Testing among BF Applications

Status	Application1	Application2	Command operator(s)	Positive/ Negative	Info
x	echo	paste	pipe: |	positive	add assumption for paste
	echo 'hello world' | paste				
x	echo	sed	substitution: ``	positive	
	echo `sed 's/hello/goodbye/' src/test/IntegrationTest/testFiles/test1.txt`				
x	paste	sed	pipe: |	positive	add assumption for paste
	echo 'hello world' | paste				

c. Pairwise Testing among EF1 Applications

Status	Application1	Application2	Command operator(s)	Positive/ Negative	Info
√	diff	grep	pipe: |	positive	
	diff src/test/IntegrationTest/testFiles/test1.txt src/test/IntegrationTest/testFiles/test2.txt | grep 'w'				
x	diff	wc	pipe: |	positive	
	diff src/test/IntegrationTest/testFiles/test1.txt src/test/IntegrationTest/testFiles/test2.txt | wc				
x	cd	wc	sequence: ;	positive	
	cd src/test/IntegrationTest/testFiles; wc -c test1.txt				
x	cd	cp	sequence: ;	positve	
	cd src/test/IntegrationTest/testFiles; cp test1.txt result.txt				

d. Pairwise Testing among EF2 Applications

Status	Application1	Application2	Command operator(s)	Positive/ Negative	Info
x	cut	ls	substitution:`` and globbing: *	positive	Found bug of ArgumentResolver
	cut -c 1 `ls src/test/IntegrationTest/testFiles/test*`				
x	ls	sort	pipe: | and globbing: *	positive	
	ls src/test/IntegrationTest/testFiles/test* | sort				
x	sort	find	pipe: | and globbing: *	positive	
	cd src/test/IntegrationTest; find testFiles -name 'test*' | sort				
x	mv	cut	sequence: ;	negative	
	mv src/test/IntegrationTest/result.txt src/test/IntegrationTest/result1.txt; cut -c 1 src/test/IntegrationTest/result.txt				

d. Pairwise Testing among other Applications

√	echo	diff	substitution: ``	positive	
	echo `diff src/test/IntegrationTest/testFiles/test1.txt src/test/IntegrationTest/testFiles/test2.txt`				
x	paste	grep	pipe: |	positive	
	paste src/test/IntegrationTest/testFiles/test1.txt | grep 'wor'				
	sed	wc	pipe: |	positive	

x	**sed 's/hello//' src/test/IntegrationTest/testFiles/test1.txt | wc -c**				
√	cd	cut	sequence: ;	positive	
	cd src/test/IntegrationTest/testFiles; cut -c 1 test1.txt				
x	cp	find	substitution: ``	positive	
	cp \`find src/test/IntegrationTest/testFiles -name 'test1.txt'\` src/test/IntegrationTest/testFiles/result.txt				

3. Testing Tools

3.1. EvoSuite:

path of generated tests: src/main/generated-test. We only save the test suite that contains failing tests.

We set 4 cores of cpu and 1 minute to generate tests for each class. In our computer, the average time to generate this test is 20 minutes.

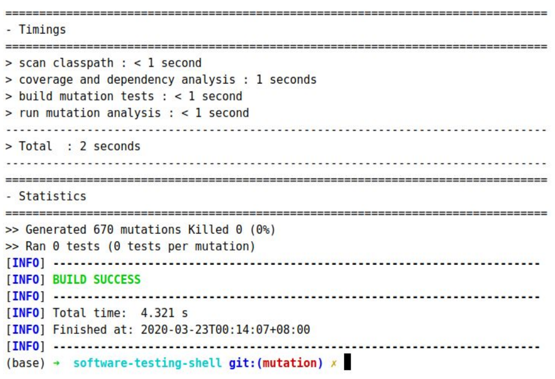
These tests failed when running in another computer. However, all the failure tests are related to different project paths in each computer.

Since evosuite will generate machine dependent tests, for example, it will change the directory which would only exist in this computer. We keep some part of tests because most of them are very good test cases and machine independent (for instance, corner test: “cd .”, diff empty string file name).

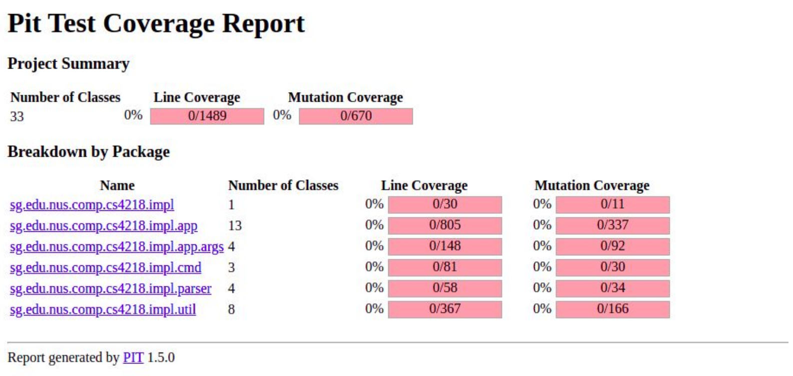
It will affect other manual tests so we do not plan to run these tests on other computers and exclude it from the test directory.

3.2. Mutation Test: PIT

We also tried PIT. However, that was not successful. It could detect tests and generate mutants, but 0 tests ran successfully. We have queried it on the internet but this problem still has not been solved.



```
================================================================================
- Timings
================================================================================
> scan classpath : < 1 second
> coverage and dependency analysis : 1 seconds
> build mutation tests : < 1 second
> run mutation analysis : < 1 second
--------------------------------------------------------------------------------
> Total  : 2 seconds
--------------------------------------------------------------------------------
================================================================================
- Statistics
================================================================================
>> Generated 670 mutations Killed 0 (0%)
>> Ran 0 tests (0 tests per mutation)
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  4.321 s
[INFO] Finished at: 2020-03-23T00:14:07+08:00
[INFO] ------------------------------------------------------------------------
(base) ➜  software-testing-shell git:(mutation) ✗ █
```



Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
33	0%	0/1489	0%	0/670

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
sg.edu.nus.comp.cs4218.impl	1	0%	0/30	0%	0/11
sg.edu.nus.comp.cs4218.impl.app	13	0%	0/805	0%	0/337
sg.edu.nus.comp.cs4218.impl.app.args	4	0%	0/148	0%	0/92
sg.edu.nus.comp.cs4218.impl.cmd	3	0%	0/81	0%	0/30
sg.edu.nus.comp.cs4218.impl.parser	4	0%	0/58	0%	0/34
sg.edu.nus.comp.cs4218.impl.util	8	0%	0/367	0%	0/166

Report generated by PIT 1.5.0