Fault Models

Halstead Checks:

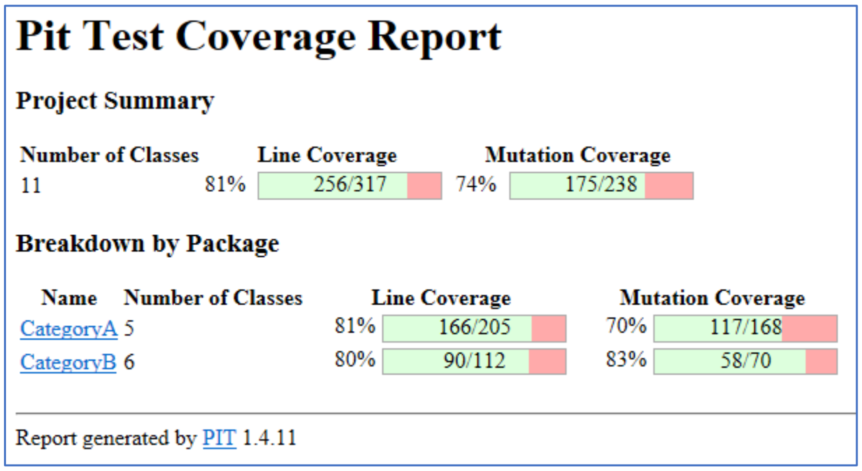
1. Difficulty
   1.1. Calculate Halstead difficulty = 0 with no operands and no unique operators
   1.2. Calculate halstead difficulty = 3 with halstead vocabulary = 2 and operands = 6
2. Effort
   2.1. Calculate halstead effort = 0 with halstead volume = 0
   2.2. Calculate halstead effort = 33 with halstead difficulty = 3 and halstead volume = 11
3. Length
   3.1. Calculate halstead length = 0 with expression with no operators/operands
   3.2. Calculate halstead length = 5 for expression: 1+2-3
4. Vocabulary
   4.1. Count 0 unique operator if the expression has no unique operators
   4.2. Count 2 unique operators if the expression has 2 unique operators
5. Volume
   5.1. Calculate Volume 0 with positive halstead length and halstead vocabulary = 0
   5.2. Calculate Volume 30 with halstead length = 10 and halstead vocabulary = 8

Number Checks:

1. Expressions
   1.1. Count 0 expressions if there are no expressions
   1.2. Count many expressions
2. Loop Statements
   2.1. Count 0 loops if file has no loops
   2.2. Count correct number of loops if file has nested loops
3. Operands
   3.1. Count 0 operands if no operands
   3.2. Count 3 operands if expression: x + y + z
4. Operators
   4.1. Count 0 operators if there are no operators
   4.2. Count 3 operators if expression: x + y – z / num
5. Single Line Comments
   5.1. Count 0 comments if the is no single line comments
   5.2. Count 1 comments if there is 1 single line comment and 1 comment block
6. Total Comments
   6.1. Count 0 comments if no comments exist
   6.2. Count 4 comment if there is 2 single line comments and 2 comment blocks

Test Improvement

White Box Testing:



Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
11	81%	256/317	74%	175/238

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
CategoryA	5	81%	166/205	70%	117/168
CategoryB	6	80%	90/112	83%	58/70

Report generated by PIT 1.4.11

Coverage improved slightly. Coverage is still missed because of not currently knowing how to test recursive calling and the logging function.

Black Box Testing:

Operator Test 1: In the operator check I was able to catch that left and right parenthesis were being counted outside of the intended range (i.e., counting parenthesis used for parameters in while loops, for loops, etc.)

Halstead Test 1: In the Halstead difficulty check if the Halstead vocabulary was zero the function would be dividing by zero and throwing an error. A check was added to the Halstead Difficulty class to catch that.

Halstead Volume Test 2: My log2() function errors out given a 0 value (divide by zero) which is correct in a math sense, but in this application, we should just make the output 0 given an input of 0.

Halstead Effort Test 2: Found my Halstead Effort class had an incorrect calculation for Halstead Volume.
Incorrect (volume = length * log2(volume))
Correct (volume = length * log2(vocabulary))

Report

Black box testing uncovered 4 bugs that white box testing didn't uncover. The coverage report wasn't really effective because there were many lines of code that I chose not to test, so it was hard to find lines of code that weren't missed on purpose. Now that I know the process of testing, I would have structured my code differently and more object oriented rather than having classes being more independent.

Class Testing

Using class testing there would be many more test cases due to the project being object oriented. Testing would take a lot longer because class testing grows very fast for each additional method added. The traditional unit testing tests individual methods where class testing tests classes between each other in their various states.

Assuming my code was ideally object oriented and I were doing class testing, I would test states of class variables when they are calling one another to make sure variables are behaving as intended.