

Nhan Nguyen  
CPTS 422  
Del3 Report  
12/8/2024

First, I needed to get Pitclipse working, so I had to fix certain issues to make that happen. During that time, I incorporated feedback from Deliverables 1 and 2 to implement fixes and improvements. This included expanding the number of Halstead operators and operands, as well as ensuring the tests passed so that Pitclipse could operate correctly.

Once Pitclipse is properly set up, here are the Pit mutation results before I officially start del3.

## Pit Test Coverage Report

### Package Summary

My422Project

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
15	100% <div><div>576/578</div></div>	86% <div><div>171/199</div></div>	87% <div><div>171/197</div></div>

### Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
<a href="#">HalsteadDifficultyCheck.java</a>	100% <div><div>69/69</div></div>	82% <div><div>14/17</div></div>	82% <div><div>14/17</div></div>
<a href="#">HalsteadEffortCheck.java</a>	100% <div><div>79/79</div></div>	73% <div><div>29/40</div></div>	73% <div><div>29/40</div></div>
<a href="#">HalsteadLengthCheck.java</a>	100% <div><div>71/71</div></div>	100% <div><div>21/21</div></div>	100% <div><div>21/21</div></div>
<a href="#">HalsteadVocabularyCheck.java</a>	100% <div><div>61/61</div></div>	44% <div><div>4/9</div></div>	44% <div><div>4/9</div></div>
<a href="#">HalsteadVolumeCheck.java</a>	97% <div><div>59/61</div></div>	33% <div><div>4/12</div></div>	40% <div><div>4/10</div></div>
<a href="#">MethodLimitCheck.java</a>	100% <div><div>12/12</div></div>	83% <div><div>5/6</div></div>	83% <div><div>5/6</div></div>
<a href="#">NumOfCommentsCheck.java</a>	100% <div><div>19/19</div></div>	100% <div><div>13/13</div></div>	100% <div><div>13/13</div></div>
<a href="#">NumOfExpressionsCheck.java</a>	100% <div><div>33/33</div></div>	100% <div><div>6/6</div></div>	100% <div><div>6/6</div></div>
<a href="#">NumOfExternalMethodReferencesCheck.java</a>	100% <div><div>31/31</div></div>	100% <div><div>18/18</div></div>	100% <div><div>18/18</div></div>
<a href="#">NumOfLinesOfCommentsCheck.java</a>	100% <div><div>20/20</div></div>	100% <div><div>12/12</div></div>	100% <div><div>12/12</div></div>
<a href="#">NumOfLocalMethodReferencesCheck.java</a>	100% <div><div>32/32</div></div>	100% <div><div>19/19</div></div>	100% <div><div>19/19</div></div>
<a href="#">NumOfLoopingStatementsCheck.java</a>	100% <div><div>14/14</div></div>	100% <div><div>6/6</div></div>	100% <div><div>6/6</div></div>
<a href="#">NumOfOperandsCheck.java</a>	100% <div><div>26/26</div></div>	100% <div><div>7/7</div></div>	100% <div><div>7/7</div></div>
<a href="#">NumOfOperatorsCheck.java</a>	100% <div><div>37/37</div></div>	100% <div><div>7/7</div></div>	100% <div><div>7/7</div></div>
<a href="#">NumOfVariableDeclarationsCheck.java</a>	100% <div><div>13/13</div></div>	100% <div><div>6/6</div></div>	100% <div><div>6/6</div></div>

After making fixes and covering missing branches, here is the Pit mutation result:

## Package Summary

### My422Project

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
15	100% <div><div>578/578</div></div>	92% <div><div>183/199</div></div>	92% <div><div>183/199</div></div>

### Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
<a href="#">HalsteadDifficultyCheck.java</a>	100% <div><div>69/69</div></div>	82% <div><div>14/17</div></div>	82% <div><div>14/17</div></div>
<a href="#">HalsteadEffortCheck.java</a>	100% <div><div>79/79</div></div>	78% <div><div>31/40</div></div>	78% <div><div>31/40</div></div>
<a href="#">HalsteadLengthCheck.java</a>	100% <div><div>71/71</div></div>	100% <div><div>21/21</div></div>	100% <div><div>21/21</div></div>
<a href="#">HalsteadVocabularyCheck.java</a>	100% <div><div>61/61</div></div>	89% <div><div>8/9</div></div>	89% <div><div>8/9</div></div>
<a href="#">HalsteadVolumeCheck.java</a>	100% <div><div>61/61</div></div>	83% <div><div>10/12</div></div>	83% <div><div>10/12</div></div>
<a href="#">MethodLimitCheck.java</a>	100% <div><div>12/12</div></div>	83% <div><div>5/6</div></div>	83% <div><div>5/6</div></div>
<a href="#">NumOfCommentsCheck.java</a>	100% <div><div>19/19</div></div>	100% <div><div>13/13</div></div>	100% <div><div>13/13</div></div>
<a href="#">NumOfExpressionsCheck.java</a>	100% <div><div>33/33</div></div>	100% <div><div>6/6</div></div>	100% <div><div>6/6</div></div>
<a href="#">NumOfExternalMethodReferencesCheck.java</a>	100% <div><div>31/31</div></div>	100% <div><div>18/18</div></div>	100% <div><div>18/18</div></div>
<a href="#">NumOfLinesOfCommentsCheck.java</a>	100% <div><div>20/20</div></div>	100% <div><div>12/12</div></div>	100% <div><div>12/12</div></div>
<a href="#">NumOfLocalMethodReferencesCheck.java</a>	100% <div><div>32/32</div></div>	100% <div><div>19/19</div></div>	100% <div><div>19/19</div></div>
<a href="#">NumOfLoopingStatementsCheck.java</a>	100% <div><div>14/14</div></div>	100% <div><div>6/6</div></div>	100% <div><div>6/6</div></div>
<a href="#">NumOfOperandsCheck.java</a>	100% <div><div>26/26</div></div>	100% <div><div>7/7</div></div>	100% <div><div>7/7</div></div>
<a href="#">NumOfOperatorsCheck.java</a>	100% <div><div>37/37</div></div>	100% <div><div>7/7</div></div>	100% <div><div>7/7</div></div>
<a href="#">NumOfVariableDeclarationsCheck.java</a>	100% <div><div>13/13</div></div>	100% <div><div>6/6</div></div>	100% <div><div>6/6</div></div>

Overall, the white-box aspect of this project is solid, or at least as solid as I could make it. While the Mutation Coverage and Test Strength are not perfect, they are close enough.

### Group 1: Halstead Metrics

Halstead metrics can face issues with miscounting operators and operands, especially in complex or less obvious scenarios like ternary expressions, lambdas, or constants. Unique operators and operands may be double-counted due to case sensitivity or scoping. Difficulty and Effort calculations can also go wrong if intermediate values like fractions are mishandled or propagated incorrectly. Errors in Length or Vocabulary often carry over into other metrics like Volume and Effort.

### Group 2: Comment Metrics

Comment metrics may fail to account for all inline comments, especially when mixed with code on the same line. Lines within multiline comments can also be miscounted, such as treating an entire block comment as a single line instead of counting each line it spans. Properly parsing and distinguishing comments from code is critical.

### Group 3: Looping and Structural Metrics

Looping and structural metrics often encounter challenges in identifying and counting nested loops or recursive structures. Special operators like method references (::) or overloaded operators in C++ are sometimes overlooked. Expressions within control structures like if or while blocks can also be missed if parsing isn't thorough, leading to undercounted metrics.

#### **Group 4: Program References and Declarations**

Counting variable declarations can be tricky if implicit declarations or shadowed variables are ignored, especially in dynamically typed languages. External method calls resolved at runtime or through reflection might be missed. Similarly, local method references invoked indirectly, such as through lambdas or callbacks, are prone to being undercounted. Properly distinguishing these elements is key to accurate metrics.

#### **Black Box Reporting**

The black box portion is definitely the most difficult part, as things get more complicated, especially with much more added files. My black box coverage is nonexistent, I believe this is because the way I set up the black box test engine led it to not communicate well with the test files. I previously thought this is due to using the same JUnit class as white box testing, but when I attempted to create a separate file for black box JUnit test, it still resulted in the same no coverage issue. After seeing no change, I did not see it was worth the time to fully make a folder of black box JUnit class testing. Oddly enough when a black box test is run, almost all of its lines run green, same with the lines in the class file as well, except for the test file, which I mentioned was the issue earlier. The part where the black box tests fail is the assertions, where all of which the actual results return zero, which indicates some odd miscommunication that I was not able to figure out. This experience has been very unfortunate, but I hope my efforts were enough.

#### **Black Box Reporting Update 12/9**

There was another attempt to get the black-box testing to work, and I'm confident the issue lies with the check classes themselves, not the Blackbox test engine code. Although the check classes work for white-box testing, they failed to work for black-box testing due to the complicated setup. I had clearly taken the wrong approach to implementing the check classes, which caused issues when attempting black-box testing.

#### **Overall JUnit Test Results**

CPTS422Project (1) (Dec 9, 2024 5:04:29 AM)					
Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions	
▼ CPTS422Project	93.3 %	8,755	629	9,384	
▼ src/test/java	90.2 %	5,800	629	6,429	
▼ My422Project.BlackBoxTestCases	0.0 %	0	626	626	
> J LocalMethodReferenceTest.java	0.0 %	0	92	92	
> J NumOfLoopsTest.java	0.0 %	0	85	85	
> J NumOfExternalMethodReferencesTest.java	0.0 %	0	70	70	
> J HalsteadLengthTest.java	0.0 %	0	67	67	
> J NumOfExpressionsTest.java	0.0 %	0	59	59	
> J NumOfOperatorsTest.java	0.0 %	0	51	51	
> J NumOfVariableOfDeclarationsTest.java	0.0 %	0	50	50	
> J HalsteadDifficultyTest.java	0.0 %	0	35	35	
> J HalsteadVocabularyTest.java	0.0 %	0	31	31	
> J HalsteadEffortTest.java	0.0 %	0	28	28	
> J NumOfCommentsTest.java	0.0 %	0	23	23	
> J HalsteadVolumeTest.java	0.0 %	0	15	15	
> J NumOfLinesOfCommentsTest.java	0.0 %	0	13	13	
> J ExternalMethodTest.java	0.0 %	0	7	7	
▼ My422Project	99.9 %	4,687	3	4,690	
> J HalsteadEffortCheckTest.java	99.5 %	638	3	641	
> J HalsteadDifficultyCheckTest.java	100.0 %	348	0	348	
> J HalsteadLengthCheckTest.java	100.0 %	483	0	483	
> J HalsteadVocabularyCheckTest.java	100.0 %	374	0	374	
> J HalsteadVolumeCheckTest.java	100.0 %	347	0	347	
> J MethodLimitCheckTest.java	100.0 %	134	0	134	
> J NumOfCommentsCheckTest.java	100.0 %	178	0	178	
> J NumOfExpressionsCheckTest.java	100.0 %	267	0	267	
> J NumOfExternalMethodReferencesCheckTest.java	100.0 %	428	0	428	
> J NumOfLinesOfCommentsCheckTest.java	100.0 %	263	0	263	
> J NumOfLocalMethodReferencesCheckTest.java	100.0 %	483	0	483	
> J NumOfLoopingStatementsCheckTest.java	100.0 %	137	0	137	
> J NumOfOperandsCheckTest.java	100.0 %	208	0	208	
> J NumOfOperatorsCheckTest.java	100.0 %	205	0	205	
> J NumOfVariableDeclarationsCheckTest.java	100.0 %	194	0	194	
▼ My422Project.BlackBoxTests	100.0 %	1,113	0	1,113	
> J HalsteadDifficultyBlackBoxTest.java	100.0 %	128	0	128	
> J HalsteadEffortBlackBoxTest.java	100.0 %	80	0	80	
> J HalsteadLengthBlackBoxTest.java	100.0 %	92	0	92	
> J HalsteadVocabularyBlackBoxTest.java	100.0 %	91	0	91	
> J HalsteadVolumeBlackBoxTest.java	100.0 %	91	0	91	
> J NumOfCommentsBlackBoxTest.java	100.0 %	79	0	79	
> J NumOfExpressionsBlackBoxTest.java	100.0 %	79	0	79	
> J NumOfExternalMethodReferencesBlackBoxTest.java	100.0 %	79	0	79	
> J NumOfLinesOfCommentsBlackBoxTest.java	100.0 %	79	0	79	
> J NumOfLocalMethodReferencesBlackBoxTest.java	100.0 %	79	0	79	
> J NumOfLoopingStatementsBlackBoxTest.java	100.0 %	78	0	78	
> J NumOfOperatorsBlackBoxTest.java	100.0 %	79	0	79	
> J NumOfVariableDeclarationsBlackBoxTest.java	100.0 %	79	0	79	
▼ src/main/java	100.0 %	2,955	0	2,955	
▼ My422Project	100.0 %	2,955	0	2,955	
> J HalsteadDifficultyCheck.java	100.0 %	408	0	408	
> J HalsteadEffortCheck.java	100.0 %	509	0	509	
> J HalsteadLengthCheck.java	100.0 %	415	0	415	
> J HalsteadVocabularyCheck.java	100.0 %	375	0	375	
> J HalsteadVolumeCheck.java	100.0 %	384	0	384	
> J MethodLimitCheck.java	100.0 %	61	0	61	
> J NumOfCommentsCheck.java	100.0 %	67	0	67	
> J NumOfExpressionsCheck.java	100.0 %	125	0	125	
> J NumOfExternalMethodReferencesCheckTest.java	100.0 %	107	0	107	
> J NumOfLinesOfCommentsCheckTest.java	100.0 %	75	0	75	
> J NumOfLocalMethodReferencesCheckTest.java	100.0 %	105	0	105	
> J NumOfLoopingStatementsCheckTest.java	100.0 %	49	0	49	
> J NumOfOperandsCheckTest.java	100.0 %	93	0	93	
> J NumOfOperatorsCheckTest.java	100.0 %	137	0	137	
> J NumOfVariableDeclarationsCheckTest.java	100.0 %	45	0	45	

## **Discussion on “Class Testing”**

Since this project isn't meant to create a usable product and is mainly focused on testing, there's very little interaction between different parts of the software. Each fault model metric is tested on its own, which makes this a great situation for class testing. Because there aren't many parts that need to work together, class testing keeps things simple.

Using class testing here can make the project cleaner and easier to manage. There would be fewer files and less code, so it's easier to focus on what each part is supposed to do. Since you're testing each metric separately, you don't need to worry about setting up complex interactions between different parts, which saves time and effort.

This approach also matches the project's purpose, which is all about making sure each metric works correctly. By testing each one on its own, you can be sure it's doing what it's supposed to without interference from other parts of the system. It also makes debugging simpler since you can easily see where a problem is coming from.

Overall, because this project has very little interaction between parts, class testing is a good fit. It keeps the project straightforward, helps avoid unnecessary complexity, and makes it easier to focus on testing each part effectively.