Justin Lawrence
WSU Fall 23
CPTS 422
Deliverable 3 Report

Fault Model

1. CommentCountCheck:
- **1.1** not including all types of comments // /**/
- **1.2** counting a system.print("/") as a comment

2. CommentLinesCountCheck:
- **2.1** counting a line more than once
- **2.2** miss counting a line

3. ExpressionCountCheck:
- **3.1** Ignoring certain types of expression X+5
- **3.2** Counting expressions more than once x=y+5-3(x*8)

4. LoopingCountCheck:
- **4.1** Not counting each type of loop
- **4.2** Not counting a nested loop

5. OperandCountCheck:
- **5.1** not including every type of operand

6. OperatorCountCheck:
- **6.1** not including all operator types

7. HalsteadDifficultyCheck:
- **7.1** incorrect formula
- **7.2** not including all operator types
- **7.3** not including every type of operand

8. HalsteadEffortCheck:
- **8.1** incorrect formula
- **8.2** not including all operator types
- **8.3** not including every type of operand

9. HalsteadLengthCheck:
- **9.1** incorrect formula
- **9.2** not including all operator types
- **9.3** not including every type of operand

10. HalsteadVocabCheck:
- **10.1** incorrect formula
- **10.2** not including all operator types
- **10.3** not including every type of operand

11. HalsteadVolumeCheck:
- **11.1** incorrect formula
- **11.2** not including all operator types
- **11.3** not including every type of operand

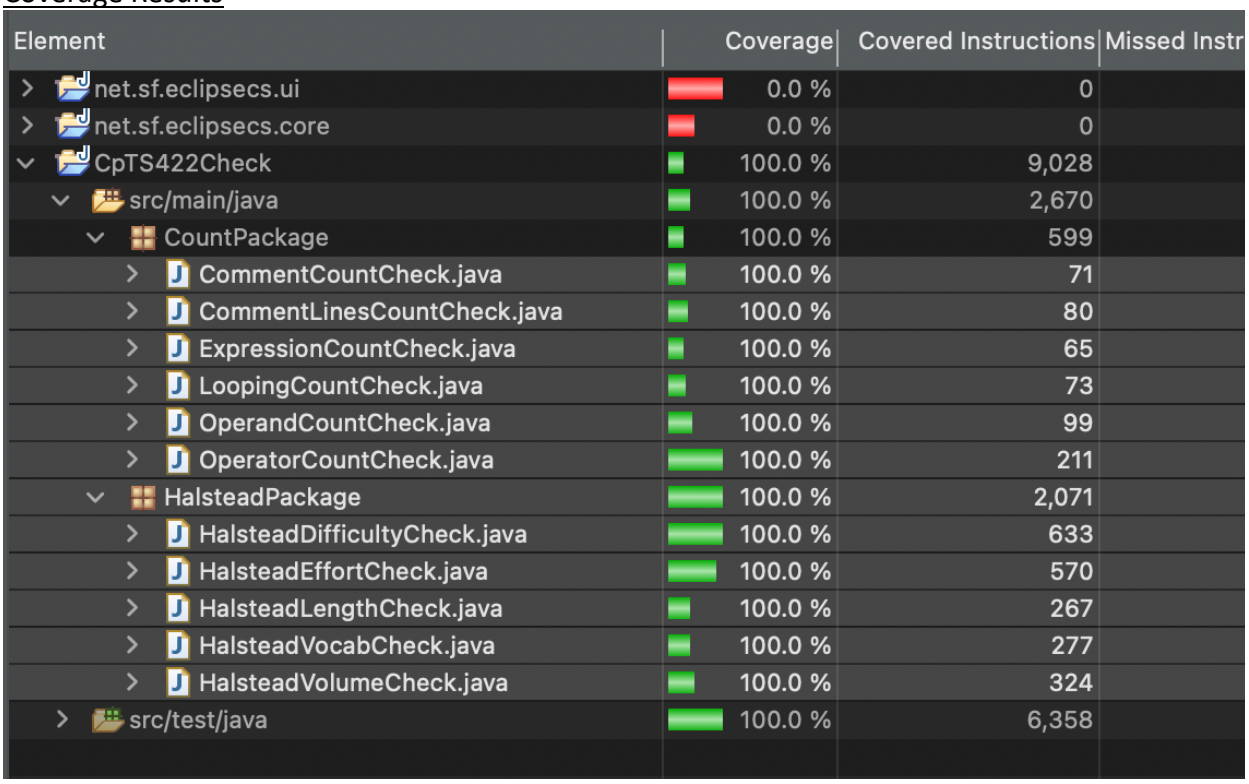
Black Box Observation

From blackbox testing my fault model I was able to see that my commentlinecheck actually didn't return the correct value. From the result I went back into my commentlinecheck and adjusted the code to return the correct value. From there I was able to check the output and then manually count the lines of comments to see if it was working correctly. I then wrote assertions inside *for(LocalizedMessage lm : check.getMessages())* to check if the messages were equal to the expected message. I started with the commentlinecheck and that passed first. Since there is only one @Test file junit only shows 1 test but being that it passed it means that all my assertions passed (I also manually did them one by one just to make sure).

Localized Message Results

```
Comment counts: 2JL
Check Done!
Comment Line Count: 7JL
Check Done!
Expression Count: 3JL
Check Done!
Looping Count: 4JL
Check Done!
Operand Count: 4JL
Check Done!
Operator Count: 4JL
Check Done!
Halstead Difficulty: 264JL
Check Done!
Halstead Effort: 19396.0JL
Check Done!
Halstead Length: 373JL
Check Done!
Halstead Vocab: 9JL
Check Done!
Halstead Volume: 1182.382025537982 6JL
Check Done!
```

Coverage Results

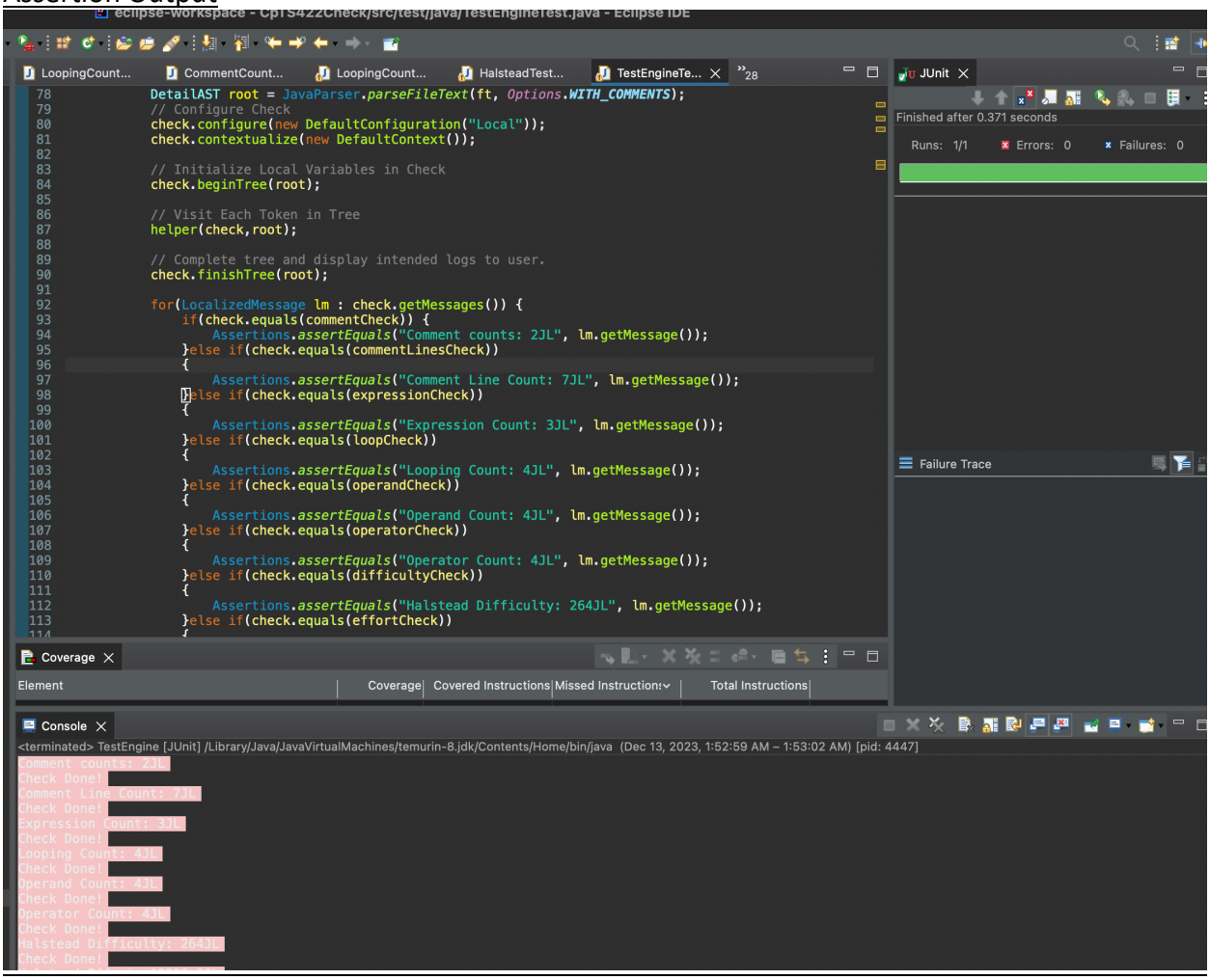


Element	Coverage	Covered Instructions	Missed Instr
net.sf.eclipsecs.ui	0.0 %	0	
net.sf.eclipsecs.core	0.0 %	0	
CpTS422Check	100.0 %	9,028	
src/main/java	100.0 %	2,670	
CountPackage	100.0 %	599	
CommentCountCheck.java	100.0 %	71	
CommentLinesCountCheck.java	100.0 %	80	
ExpressionCountCheck.java	100.0 %	65	
LoopingCountCheck.java	100.0 %	73	
OperandCountCheck.java	100.0 %	99	
OperatorCountCheck.java	100.0 %	211	
HalsteadPackage	100.0 %	2,071	
HalsteadDifficultyCheck.java	100.0 %	633	
HalsteadEffortCheck.java	100.0 %	570	
HalsteadLengthCheck.java	100.0 %	267	
HalsteadVocabCheck.java	100.0 %	277	
HalsteadVolumeCheck.java	100.0 %	324	
src/test/java	100.0 %	6,358	

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
(default package)	100.0 %	6,358	0	6,358
CommentCountTest.java	100.0 %	191	0	191
CommentCountTest	100.0 %	191	0	191
testBeginTree()	100.0 %	19	0	19
testFinishTreeNormal()	100.0 %	43	0	43
testFinishTreeNull()	100.0 %	31	0	31
testGetAcceptableTokens()	100.0 %	20	0	20
testGetDefaultTokens()	100.0 %	20	0	20
testGetRequiredTokensTest()	100.0 %	20	0	20
testIsCommentNodesRequired()	100.0 %	11	0	11
testVisitToken()	100.0 %	19	0	19
CommentLinesCountTest.java	100.0 %	215	0	215
CommentLinesCountTest	100.0 %	215	0	215
testBeginTree()	100.0 %	19	0	19
testFinishTreeNormal()	**100.0 %**	**43**	**0**	**43**
testFinishTreeNull()	100.0 %	31	0	31
testGetAcceptableTokens()	100.0 %	28	0	28
testGetDefaultTokens()	100.0 %	28	0	28
testGetRequiredTokensTest()	100.0 %	28	0	28
testIsCommentNodesRequired()	100.0 %	11	0	11
testVisitToken()	100.0 %	19	0	19
ExpressionCountTest.java	100.0 %	168	0	168
ExpressionCountTest	100.0 %	168	0	168
testBeginTree()	100.0 %	19	0	19
testFinishTreeNormal()	100.0 %	43	0	43
testFinishTreeNull()	100.0 %	31	0	31
testGetAcceptableTokens()	100.0 %	16	0	16
testGetDefaultTokens()	100.0 %	16	0	16
testGetRequiredTokensTest()	100.0 %	16	0	16
testVisitToken()	100.0 %	19	0	19
HalsteadDifficultyTest.java	100.0 %	982	0	982
HalsteadDifficultyTest	100.0 %	982	0	982
testBeginTree()	100.0 %	19	0	19
testFinishTreeWhenNotZero()	100.0 %	45	0	45
testFinishTreeWhenZero()	100.0 %	43	0	43
testGetAcceptableTokens()	100.0 %	236	0	236
testGetDefaultTokens()	100.0 %	236	0	236
testGetOperandTotalTest()	100.0 %	102	0	102
testGetRequiredTokensTest()	100.0 %	236	0	236
testVisitTokenOperand()	100.0 %	19	0	19
testVisitTokenOperator()	100.0 %	38	0	38

ement	Coverage	Covered Instructions	Missed Instructions	Total Instructions
testVisitTokenOperator()	100.0 %	38	0	38
HalsteadEffortTest.java	100.0 %	982	0	982
HalsteadEffortTest	100.0 %	982	0	982
testBeginTree()	100.0 %	19	0	19
testFinishTreeWhenNotZero()	100.0 %	45	0	45
testFinishTreeWhenZero()	100.0 %	43	0	43
testGetAcceptableTokens()	100.0 %	236	0	236
testGetDefaultTokens()	100.0 %	236	0	236
testGetOperandTotalTest()	100.0 %	102	0	102
testGetRequiredTokensTest()	100.0 %	236	0	236
testVisitTokenOperand()	100.0 %	19	0	19
testVisitTokenOperator()	100.0 %	38	0	38
HalsteadLengthTest.java	100.0 %	797	0	797
HalsteadLengthTest	100.0 %	797	0	797
testBeginTree()	100.0 %	19	0	19
testFinishTree()	100.0 %	43	0	43
testGetAcceptableTokens()	100.0 %	236	0	236
testGetDefaultTokens()	100.0 %	236	0	236
testGetRequiredTokensTest()	100.0 %	236	0	236
testVisitToken()	100.0 %	19	0	19
HalsteadVocabTest.java	100.0 %	797	0	797
HalsteadVocabTest	100.0 %	797	0	797
testBeginTree()	100.0 %	19	0	19
testFinishTree()	100.0 %	43	0	43
testGetAcceptableTokens()	100.0 %	236	0	236
testGetDefaultTokens()	100.0 %	236	0	236
testGetRequiredTokensTest()	100.0 %	236	0	236
testVisitToken()	100.0 %	19	0	19
HalsteadVolumeTest.java	100.0 %	1,112	0	1,112
HalsteadVolumeTest	100.0 %	1,112	0	1,112
testBeginTree()	100.0 %	19	0	19
testFinishTreeWhenNotZero()	100.0 %	45	0	45
testFinishTreeWhenZero()	100.0 %	43	0	43
testGetAcceptableTokens()	100.0 %	236	0	236
testGetDefaultTokens()	100.0 %	236	0	236
testGetLengthTest()	100.0 %	270	0	270
testGetRequiredTokensTest()	100.0 %	236	0	236
testVisitToken()	100.0 %	19	0	19
LoopingCountTest.java	100.0 %	192	0	192
OperandCountTest.java	100.0 %	293	0	293
OperatorCountTest.java	100.0 %	629	0	629
LoopingCountTest.java	100.0 %	192	0	192
LoopingCountTest	100.0 %	192	0	192
testBeginTree()	100.0 %	19	0	19
testFinishTreeNormal()	100.0 %	43	0	43
testFinishTreeNull()	100.0 %	31	0	31
testGetAcceptableTokens()	100.0 %	24	0	24
testGetDefaultTokens()	100.0 %	24	0	24
testGetRequiredTokensTest()	100.0 %	24	0	24
testVisitToken()	100.0 %	19	0	19
OperandCountTest.java	100.0 %	293	0	293
OperandCountTest	100.0 %	293	0	293
testBeginTree()	100.0 %	19	0	19
testFinishTree()	100.0 %	43	0	43
testGetAcceptableTokens()	100.0 %	68	0	68
testGetDefaultTokens()	100.0 %	68	0	68
testGetRequiredTokensTest()	100.0 %	68	0	68
testVisitToken()	100.0 %	19	0	19
OperatorCountTest.java	100.0 %	629	0	629
OperatorCountTest	100.0 %	629	0	629
testBeginTree()	100.0 %	19	0	19
testFinishTree()	100.0 %	43	0	43
testGetAcceptableTokens()	100.0 %	180	0	180
testGetDefaultTokens()	100.0 %	180	0	180
testGetRequiredTokensTest()	100.0 %	180	0	180
testVisitToken()	100.0 %	19	0	19

Assertion Output



Mutation Coverage Output

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
15	68%	235/347	33%	39/118	64%	39/61

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage	Test Strength
BlackBoxPackage	4	0% 0/78	0% 0/25	0% 0/0
CountPackage	6	81% 76/94	30% 12/40	55% 12/22
HalsteadPackage	5	91% 159/175	51% 27/53	69% 27/39

Report generated by PIT 1.6.8

Mutation Observation

From the mutation testing results it looks like I could benefit from testing a greater variety of variations to potentially catch different bugs. As far as line coverage goes my packages were adequately covered but it appears after the mutations my line coverage dropped from the initital 100% from DEL 2. To counter this I can create more targeted tests in the area where it dropped substantially, in this case CountPackage. To improve testing strength I could create a great variety of tests such as exception handling, and boundary testing.

Class Testing Observation

Class testing is useful for isolating individual units of code to ensure that objects are behaving as intended such as the state of the object. However, it is not possible, (as discussed in class) to test every possible sequence, so class testing needs to be done in a focused manner to test specific units of code. Class testing is also useful to test inheritance by testing only the new parts of the derived class. So for this project if we had a base class Halstead for instance, and we created a class test for the base then for all the derived Halstead classes such as length, volume, difficulty, etc. we would only have to test the new additions and not have to retest everything from the base which could save time, making testing more efficient.