Arlo Jones

12/8/2022

White box testing:

All branches are 100% covered by the white box tests, as was the case in milestone 2.

To conduct black box test, an additional function was added to the project, getResults(). A test was added to ensure this was covered as well and 100% branch coverage was maintained.

Using pitClipse to run mutation test returned the following results:

Table

Description automatically generated

The only bugs that were not squashed had to do with the log function of the checks.

Black box testing:

For black box testing, the following fault models were used:

1. Expressions
   1. No expressions
   2. Unary operators
   3. Parentheses could create subexpressions that are over counted
   4. Parsing of operators like += or ++ might not count the implied expression
2. Halstead Difficulty
   1. Incorrect metric equation
   2. Divide by zero (Should be impossible with legitimate code)
   3. Repeat tokens (could fail to increment counts appropriately if it isn’t identified as duplicate)
3. Halstead Effort
   1. Incorrect metric equation
   2. Incorrect logarithm
   3. Divide by zero (Should be impossible with legitimate code)
   4. Repeat tokens (could fail to increment counts appropriately if it isn’t identified as duplicate)
4. Halstead Length
   1. Token being counted as both operator and operand
   2. Nested functions
5. Halstead Operands
   1. Declarations counting when they shouldn’t
   2. Operands in unary expressions
   3. Parameters
   4. Function definitions
6. Halstead Operators
   1. Modifiers / type declarations being counted
   2. Both sides of pairs (things like {},()) being counted
   3. Operators in for loops
   4. Nested function calls
   5. Nested if-else
7. Halstead Vocabulary
   1. Token being counted as both operator and operand
   2. Different instance of the same token counting twice
8. Halstead Volume
   1. Incorrect metric equation
   2. Incorrect logarithm
   3. Repeat tokens (could fail to increment counts appropriately if it isn’t identified as duplicate)
9. Lines of comments
   1. No comments
   2. Block comments
   3. Inline comments
10. Looping statements
    1. No loops
    2. Not all loop types accounted for
    3. For each
    4. Do while counting twice
    5. Nested loops
11. Number of comments
    1. No comments
    2. Block comments
    3. In line comments
    4. Two blocks on the same line

One bug that was caught by the black box tests where with the lines of comments check. I had assumed that the comment content token would register per line of comment in multiline comments, but it does not. Because of this, both number of comments and lines of comments were being calculated the same way. I adjusted lines of comments to instead calculate the line number difference between the beginning of the multiline comment block and then end. This resulted in accurate counting but was only discovered by a test case that involved a multi-line comment.

All black box tests pass.

Class testing:

The biggest thing with class testing is that the expected response from a given function within a class likely changes based upon the state of the class when the function is called

Another important consideration with class testing, are the relationships between classes as they call functions in each other.

For the checks I’ve written, there is generally little state preserved between function calls save from an ongoing tally of collected data, that is then parsed and calculated in finishTree() if it needs to be. So the expected response from visitToken() is to update the sate of the check, which in turn modifies the output of finishTree() when it’s called. It would not be feasible to test every (state, function call) pair in a given check, but testing boundary values, or values that might cause faults in the finishTree() calculation of a check would be a good start.

Additionally, there are some checks for which the type of token called in visit token impacts the state of the check. Testing the internal state of a check is not as straight forward as unit testing, but verifying that those interaction correctly update the state would be important. This lends itself well to category partition testing, because the logic for how to update the state depending on the type of token breaks up the tokens into distinct subsets, and so the logic can be tested using one representative from each subset.

Lastly, the checks do interact with a class that feeds them token lists. Because the interactions there are purely to get information and there is no modification of the data, the state function pairs don’t have an impact.