Lab 3 Report

Lab Report #3  
ECE 322  
Yenosibina Makanjuola 1629453

Introduction (3 marks)

The purpose of this laboratory assignment is to become familiar with the rudimentary techniques of white-box testing, specifically unit testing. The main goal of the problem is to identify faults in the program that has been given to us using both unit testing, pairwise testing, and any other testing techniques learned in the course. The methodologies used include **statement coverage**, **branch coverage**, and **unit testing.**

The lab involves testing functions in the different programs to ensure statement and branch coverage, the testing methodologies are implemented using the unittest framework in python which is a tool that makes testing easier. Statement coverage ensures that every line of code is executed at least once, while branch coverage verifies that all possible branches in conditional statements are executed.

Lastly pairwise testing is used to effectively cover all input combinations without having to test every possible input combination. The use of these methodologies and frameworks creates a comprehensive approach in finding errors and defects in the programs. [1]

# Part One –General (5 marks)

**Q1** : First, identify Unit testing, and describe the purposes of unit testing as a white box testing methodology. Why we use unit testing?

**Answer**: Unit testing is a software testing technique that verifies whether each component or function in an application works well in isolation from the entire application. It is used as a white box testing technique because it allows the developer to test the internal logic and correctness of the program because unit test can’t be written without an understanding of the program. Unit tests are used because they ensure the reliability of components and reduce testing cost in later stages. [2]

**Q2** : Explanation of what code coverage criteria are, why they are important, and what statement and branch coverage are specifically?

**Answer**: Code coverage criteria are metric used to measure how thoroughly the code is tested. This ensures comprehensive testing and improves code quality. Statement coverage ensures that every line of code is executed at least once, while branch coverage ensures each possible branch(conditional statement) in the code has been executed. [3]

**Q3** : Explanation of the purpose of pairwise testing and its benefits

**Answer**:

Pairwise testing considers a set of test cases that “covers” all combinations of the test data for each pair of variables. Pairwise testing greatly reduces the number of test cases [4]

**Q4** : Explanation of the purpose of using Mock objects in Testing

**Answer**: Mock objects are used to isolate the specific unit of code being tested and created an ideal environment in which consistent and reliable results can be produced.

# Part One –Bisect (40 marks)

**Q1** : Description of the problem required, identification of what the program is supposed to do in this case:

**Answer**: In this part of the lab we are asked to use white box testing techniques to test the mybisect program. The test cases generated should ensure statement coverage, branch coverage, and ensure that the while loop is executed at least twice. The program under testing enables us to find the root of a function of format f(x) = 0 within a given interval.

**Q2**: Control flow graph diagram for the algorithm

**Answer**:

A line of circles and lines

Description automatically generated

**Q3**: Test case tables

* + actual values for test cases and actual results are required
  + Stating that the result is a pass is acceptable if the description and expected value are clear.
  + Description should identify what the test is for, what it tests.

**Answer**:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Id | Description | Tolerance | Max Iterations | Function | X1 | X2 | Expected Result | Actual Result |
| 1 | Testing the constructor with all 3 arguments in the function given(Tolerance , Max Iterations , and Function)  f = Polynomial(1 , -1)  b = MyBisect(0.0001 , 100 , f) | 0.0001 | 100 | f(x) = x - 1 | - | - | b.tolerance  = 0.0001  b.max\_iterations = 100  b.func = x - 1 | b.tolerance  = 0.0001  b.max\_iterations = 100  b.func = x - 1 |
| 2 | Testing the constructor with tolerance and function  f = Polynomial(1 , -1)  b = MyBisect(0.0001 , f) | 0.0001 | - | f(x) = x - 1 | - | - | b.tolerance  = 0.0001  b.max\_iterations = 50  b.func = x - 1 | b.tolerance  = 0.0001  b.max\_iterations = 50  b.func = x - 1 |
| 3 | Testing the constructor with maxIterations and function  f = Polynomial(1, -1) b = MyBisect(100, f) | - | 100 | f(x) = x -1 | - | - | b.tolerance  = 0.000001  b.max\_iterations = 100  b.func = x - 1 | b.tolerance  = 0.000001  b.max\_iterations = 100  b.func = x - 1 |
| 4 | Testing the constructor with just function  f = Polynomial(1, -1)  b = MyBisect(f) | - | - | f(x) = x -1 | - | - | b.tolerance  = 0.000001  b.max\_iterations = 50  b.func = x - 1 | b.tolerance  = 0.000001  b.max\_iterations = 50  b.func = x - 1 |
| 5 | Testing the constructor with  no arguments | - | - | - | - | - | b.tolerance  = 0.000001  b.max\_iterations = 50  b.func = None | b.tolerance  = 0.000001  b.max\_iterations = 50  b.func = None |
| 6 | Testing the setters of the max iteratioins and setters of the tolerance when they are within the accepted range | 0.0001 | 100 | f(x) = x -1 | - | - | b.tolerance and b.max\_iterations has been set to the assigned value | b.tolerance and b.max\_iterations has been set to the assigned value |
| 7 | Testing the setters of the max iteratioins and setters of the tolerance when they are outside the accepted range | 0.0001 | 100 | f(x) = x -1 | - | - | b.tolerance and b.max\_iterations remain the same value | b.tolerance and b.max\_iterations remain the same value |
| 8 | Testing whether the root can be found | 0.000001 | 50 | f(x) = x - 1 | 1 | -1 | 1(Approximately) | 1(Approximately) |
| 9 | Testing function that does not have real roots | 0.000001 | 50 | f(x) = x^2 -1 | -1 | 1 | “Root not Found” | “Root not found” |
| 10 | Testing convergence by tolerance | 0.001 | 50 | f(x) = x^2 -4 | 1 | 3 | 2(Approximately) | 2(Approximately) |
| 11 | Test run function exceeding the maximum number of iterations | 0.000001 | 5 | f(x) = x^3 -2x + 2 | -2 | 2 | “Root not Found” | “Root not Found” |
| 12 | Test the run function with multiple iterations | 0.000001 | 50 | f(x) = x^2 - 9 | 2 | 5 | 3(Approximately) and multiple iterations took place | 3(Approximately) and multiple iterations took place |
| 13 | Test run function with no function given | 0.000001 | 50 | - | 2 | 5 | Error message -”No function given” | Program crashed |
| 14 | Test the run function with negative values initialized for the tolerance and max iterations | -0.0001 | -50 | f(x) = x^2 - 9 | 2 | 5 | “Root not Found” | “Root not Found” |

**Q4**: Discussion Effectiveness of statement coverage, how hard is it to attain?

* + Effectiveness of branch coverage, how hard is it to attain?
  + Did the tests achieve these criteria?
  + Where any errors discovered?
  + How difficult would path testing be?

[6]

**Answer**:

Statement Coverage: Statement coverage ensures that every line of program is executed at least once, and it is effective in finding syntax errors and logical errors but it is not effective in verifying the correctness of conditional logic and testing how different input combinations affect the output. Statement coverage is easy to attain because it just needs to be ensured that every line of code runs.

Branch Coverage: Branch coverage ensures every possible branch in the code is executed at least once. Branch coverage is more effective than statement coverage in revealing logical errors. Branch coverage could be hard to attain depending on the complexity of the code and the number of conditional statements.

Errors Discovered: Yes, there was an error discovered. Test case 13 failed because no function was given, and the user tried to find the roots of a function that does not exist.

Path Testing: Path testing would be very difficult to attain as we would have to consider every possible combination of execution paths in the code which is very cumbersome to come up with.

**Q5** : Check the code for the stated coverage and HTML coverage report :

**Answer**:

Your test code files and your HTML coverage report

Test code and HTML coverage included in file.

# Part 2 Mocking (35)

**Q1**: Test case tables

* + actual values for test cases and actual results are required
  + Stating that the result is a pass is acceptable if the description and expected value are clear.
  + Description should identify what the test is for, what it tests.

**Answer**:

[7]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Description** | **Input** | **Expected Output** | **Actual Output** | **Result** |
| TC1 | Test random function for correct length and range | MathPackage.random(5, 6, 10) | Array of length 5, values between 6 and 10 | Array of length 5, values between 6 and 10 | Pass |
| TC2 | Test random function with zero length | MathPackage.random(0, 6, 10) | Empty array | Empty array | Pass |
| TC3 | Test random function with negative range | MathPackage.random(7, -7, -4) | Array of length 7, values between -7 and -4 | Array of length 7, values between -7 and -4 | Pass |
| TC4 | Test random function with boundaries switched | MathPackage.random(7, 10, 6) | Raise ValueError | No Value error raised values still generated | Fail |
| TC5 | Mock test for random function output | Mocked MathPackage.random(3, 3.5, 5.6) | [3.6, 3.8, 5] | [3.6, 3.8, 5] | Pass |
| TC6 | Test max function with empty list | MathPackage.max([]) | Raise ValueError | sys.float\_info.min in python is returned | Fail |
| TC7 | Test max function with single element | MathPackage.max([1]) | 1 | 1 | Pass |
| TC8 | Test max function with positive numbers | MathPackage.max([1, 2, 3]) | 3 | 3 | Pass |
| TC9 | Test max function with positive and negative numbers | MathPackage.max([-1, 2, -3]) | 2 | 2 | Pass |
| TC10 | Test max function with negative numbers | MathPackage.max([-1, -2, -3]) | -1 | 2.2250738585072014e-308 | Fail |
| TC11 | Test max function with zero | MathPackage.max([0, -2, -3]) | 0 | 2.2250738585072014e-308 | Fail |
| TC12 | Mock test for max function | Mocked MathPackage.max([1, 6, 5]) | 6 | 6 | Pass |
| TC13 | Test min function with empty list | MathPackage.min([]) | Raise ValueError | sys.float\_info.max | Fail |
| TC14 | Test min function with single element | MathPackage.min([1]) | 1 | 1 | Pass |
| TC15 | Test min function with positive numbers | MathPackage.min([1, 2, 3]) | 1 | 1 | Pass |
| TC16 | Test min function with negative numbers | MathPackage.min([-1, -2, -3]) | -3 | -3 | Pass |
| TC17 | Test min function with positive and negative numbers | MathPackage.min([-1, 2, -3]) | -3 | -3 | Pass |
| TC18 | Test min function with zero | MathPackage.min([1, 0, 3]) | 0 | 0 | Pass |
| TC19 | Mock test for min function | Mocked MathPackage.min([6, 7, 8]) | 6 | 6 | Pass |

**Q2**: explain why we need mocking in Testing and comment on how effective it is.

**Answer**:

# Mock testing is essential because it allows testers to isolate specific parts of the code and concentrate on verifying the core logic without being influenced by external factors that could affect the code’s behavior. Mock testing is very effective because it allows the tester to create a controlled environment that produces consistent and reliable results. [8]

**Q3** : Check the code for the stated coverage and HTML coverage report :

**Answer**:

Your test code files and your HTML coverage report

Test code and HTML coverage included in file.

# Part 3 – Pairwise (15 marks)

**Q1**: An orthogonal array, close to matching problem or exactly matching problem

**Answer**:

An orthogonal array can be either close to the matching problem or exactly the matching problem depending on the problem that is being tested. If the orthogonal array perfectly fits the requirements then it would match but if not there would be a need for approximations.

**Q2**: List the programmatically generated tests required

* + Should exactly match the specification
  + Should identify what program was used to generate the cases

**Answer**:

The test cases needed are shown below and the program used to generate the test cases is AllPairs from allpairspy.

A screenshot of a black screen

Description automatically generated

**Q3**: Discussion

* + Comparison of program to orthogonal arrays, which resulted in fewer test cases? Which is easier? Etc
  + Effectiveness of these types of tools? Reduction of the number of test cases?
  + Discussion of what pairwise testing tests, how does it catch errors?
  + Discussion of strengths and weaknesses of this method

[9]

**Answer**:

1. Comparison of Orthogonal Arrays: The number of test cases need to be generated for the problem would normal be 27 due to the fact that each variable has 3 possible outcomes. The use of orthogonal arrays and the tool used (Allpairspy) greatly reduced the number of test cases. Orthogonal arrays are more accurate than Allpairspy since the tool uses approximations
2. Effectiveness of Tools used: Allpairspy is effective for generating a minimal set of testcases that cover pairwise combinations which saves time. The effectiveness of this tool is dependent on the usage, if there are high order interaction between a lot of variables it does not necessary cover all the interactions and there may still be bugs.
3. Discussion of Pairwise Testing: Pairwise testing considers a set of test cases that “covers” all combinations of the test data for each pair of variables. Pairwise testing catches errors by ensuring that all possible pairs of input values are tested, uncovering defects that arise from interactions between two variables, which account for most common software bugs.
4. Weaknesses and Strengths of Pairwise Testing:
   1. Weaknesses –
      1. Pairwise testing does not guarantee detection of bugs caused by interactions involving three or more variables, so it may miss complex defects that require multiple variables interacting simultaneously
      2. It is not designed to catch issues that occur only in specific sequences or under time-sensitive conditions, as it focuses solely on variable combinations rather than order or timing.
   2. Strengths –
      1. Significantly reduces the number of test cases
      2. Effective for finding interaction faults

**Q4**: Code

**Answer**: pairwise code

# Conclusion and discussion (2 marks)

In this lab we were able to detect errors in the program using white box testing techniques, such as unit and pairwise testing. In the Mathpackage program all the min and max functions returned the wrong value when an empty list was given due to the implementation already initializing a max and min value, also the max function does not work with negative values due to this same implantation. The application of statement and branch coverage ensured that all the programs were tested thoroughly, while the use of pairwise testing and orthogonal arrays provide an efficient wat of testing the program. Mock testing was implemented in the test cases to give a general idea of how it works. While pairwise testing is effective in identifying interaction-based errors, it is limited in its ability to catch multi-variable dependencies, highlighting the trade-offs between efficiency and completeness. Overall, this lab demonstrates how these testing strategies contribute to code quality, with observations on their effectiveness, limitations, and practical applications. [10]

**References**

[1] – Chatgbt (Prompt – Help me adjust my introduction to match the requirements specified)

[2] – Chatgbt (Prompt - Question provided to ChatGPT for opinion)

[3] – Chatgbt (Prompt - Question provided to ChatGPT for opinion)

[4] – Lab Manual

[6] – Chatgbt (Prompt - Question provided to ChatGPT for opinion)

[7] – Chatgbt (Prompt -Generate test case table from test cases)

[8] – Chatgbt (Prompt - Question provided to ChatGPT for opinion)

[9] – Chatgbt (Prompt – Questions provided to ChatGPT for opinion)

[10] – Chatgbt (Prompt – Help me adjust my conclusion to match the requirements specified)