DIT636 / DAT560 - Assignment 4: Mutation Testing

Due Date: Sunday, March 8th, 23:59 (Via Canvas)

There are three questions worth a total of 100 points. You may discuss these problems in your teams and turn in a single submission for the team (consisting of a zipped archive containing a PDF report and test code) on Canvas. Answers must be original and not copied from online sources or generated using AI tools.

Report Template: A template for your report can be found here. You may modify this template for your own purposes, if needed.

Cover Page: On the cover page of your report, include the name of the course, the date, your group name, and a list of your group members.

Peer Evaluation: All students must also submit a peer evaluation form. This is a separate, individual submission on Canvas.

Overview

In this question, you will manually apply Mutation Testing to the Smart Pet Feeder example from Assignment 3 and assess the test cases you created as part of that assignment.

The source code is available from
https://github.com/EsmeYi/dit636_examples/tree/main/as2-petfeeder

You should work with the version of the code where you have fixed detected faults. This will make it easier to assess whether tests fail due to a mutation or a real fault.

Problem 1 - Mutant Creation (64 Points)

Create eight mutants for classes from the Smart Pet Feeder project. The mutants should be spread among the following four categories:

- One mutant must be **invalid** (does not compile).
- One must be **equivalent** to the original code (you inserted a fault, but no test case can possibly yield a different solution to the original code).
- Three mutants must be **valid-but-not-useful** (all tests, or almost all tests, will expose this mutation).
- Three mutants must be **useful** (only a small number of specifically-designed tests will expose this mutation).

You must apply at least four different mutation operators across this set.

Each mutant should apply exactly one operator, and be distinct from the other mutants (e.g., if you apply arithmetic operator replacement to the same location twice, changing to a + and a *, those mutants are not distinct enough).

The mutation operators can be drawn from the attached handout or the Lecture 11 slides. For more information on the mutation operators, see Chapter 16 of Software Testing and Analysis.

Hint: You do not have to use the same classes or methods for all mutant categories. Try mutating different parts of the code. You may use any class except Main or the two exception classes.

Your report should include, for each mutant:

- The original and mutated code.
- The mutation operator applied.
- An explanation of how the mutant differs from the original code
- Whether it is invalid, equivalent, valid-but-not-useful, or useful.
- An explanation and why it belongs in that category.
- If not equivalent, an explanation of how the mutant can be detected. Feel free to use example input to illuminate this explanation.

Points are divided as follows: 8 points per mutant, with 2 points for each explanation and 1 point for each of the other requested items.

Problem 2 - Mutant Assessment (26 Points)

In this problem, you will assess the test suite that you created in Assignment 3 in terms of the test suite's ability to detect the non-equivalent mutants created in the previous problem.

In your report:

- Identify which test cases expose which of your mutants (test cases that expose a mutant pass on the original code and fail on the mutated code). **(10 Points)**.
- Explain why these tests expose each mutant. For any non-equivalent mutants not exposed, explain why they were not exposed by your existing test suite. **(16 Points)**

Problem 3 - Test Suite Expansion (10 Points)

If needed, design additional test cases that detect the remaining mutants.

- Describe each new test case.

    - Describe why that test detects each new mutant that it detects, highlighting what differentiates the new test from past test cases.
- If all mutants were detected by existing tests (i.e., no new tests were needed), then explain this in your answer.

Include both the report and the code for any new test cases in the zip file you submit.