Software Testing Report

Ahmet Bekir Arslanalp (210717012)

April 11, 2025

1. Overview

This report presents the structural and mutation testing results of the `ArrayUtils` class developed as part of the `lab01` project. The goal of this evaluation is to determine how well the written test cases cover the code logic and how effectively they catch potential defects.

Two different tools were used during the testing phase:

- **JaCoCo** – a code coverage tool that measures how much of the source code is executed during testing.
- **PIT (Pitest)** – a mutation testing tool that introduces small changes (mutants) into the code to check if the test cases can detect them.

2. Code Coverage Results (JaCoCo)

The code coverage metrics were generated using JaCoCo and exported to a CSV file (`jacoco.csv`). The results are shown in the table below:

Table 1: Code Coverage (JaCoCo)

Metric	Missed	Covered	Total	Coverage %
Instructions	3	84	87	96.55%
Branches	0	10	10	100.00%
Lines	1	18	19	94.74%
Complexity	1	7	8	87.50%
Methods	1	2	3	66.67%

Overall, the structural coverage is strong, particularly in branch and instruction metrics. This indicates that the test cases execute most of the important paths in the code.

3. Mutation Testing Results (PIT)

Mutation testing was done using PIT version 1.5.1 with a wide set of mutators enabled. The tool generates mutants by making small changes to the source code and checks if the tests fail as a result.

Below is the summary of mutation results:

- **Total Mutations:** 260
- **Killed Mutants:** 129
- **Survived Mutants:** 131
- **Mutation Coverage:** 49.6% $\approx$ **50%**

Table 2: Mutation Test Summary

Metric	Count	Percentage
Total Mutants	260	100%
Killed Mutants	129	49.6%
Survived Mutants	131	50.4%

4. Observations and Analysis

- High coverage in JaCoCo (e.g. 100% branch coverage) shows that tests are running all logic branches.
- However, mutation testing provides a deeper look into the strength of those tests. In this case, only half of the mutants were detected by the current tests.
- Some of the survived mutations are likely caused by print statements, constant replacements, or simple operations that do not affect program behavior visibly.
- The logic inside the `findLastIndex` method is relatively simple, so certain changes may not result in a different output unless very specific edge cases are tested.
- The tests could be extended by adding more assertions and covering scenarios such as boundary values, null arrays, or unusual inputs.

5. Conclusion

While JaCoCo shows a high level of structural coverage, mutation testing highlights potential weaknesses in the current test suite. Even though 96% instruction and 100% branch coverage were achieved, only 50% of the mutants were killed.

For the current code, this mutation score is acceptable. The `ArrayUtils` class is simple and does not involve complex business logic or critical state changes. Many of the mutations involve harmless alterations that would not necessarily trigger test failures, such as changing loop counters or modifying console output.

Nevertheless, mutation testing was valuable in identifying gaps. It showed that writing tests is not just about coverage percentages, but also about verifying meaningful behavior. With a few more tests targeting specific inputs and edge cases, the mutation score could be improved even further.