**Software Testing Project**

*Student: Tung Lam Nguyen*

*Instructor: Dr. Alex Groce*

*Course: CS567*

*Semester: Fall 2023*

**1. Introduction**

This project explores mutation testing based on insights from the CS567 course. The report presents personal findings to provide a practical understanding of mutation testing. Two mutators, mutpy (last updated in 2020 but compatible with Python 3 due to minimal language changes) and universalmutator, were employed to assess their impact on Python code. Testing involved using both the unittest framework and tstl.

Project GitHub: <https://github.com/lamtung16/CS567_Project_Software_Testing>

mutpy: <https://pypi.org/project/MutPy/>

universalmutator: <https://pypi.org/project/universalmutator/>

tstl: <https://pypi.org/project/tstl/>

**1.1 Understanding Mutation Testing**

Mutation testing checks how well a test suite works by purposefully introducing mistakes into the code and seeing if the suite can spot these errors. For example, changing an addition to a subtraction in a simple function act as a way to check how thorough the test suite is. This helps determine if the testing process is effective in finding and dealing with potential code problems.

**1.2 Tools Utilized**

The project uses mutpy and universalmutator as tools to generate mutants, while tstl and unittest are employed for testing. These tools provide various features and capabilities, enabling a thorough exploration of mutation testing in Python.

**1.3 Objectives of the Study**

Instead of declaring a winner, the project aims to offer a nuanced comparison between mutpy and universalmutator. The report provides valuable insights into the practical applications of mutation testing tools. The following sections explore the experimental process and results, presenting a comprehensive overview of the dynamic mutation testing landscape.

**2. Experiments**

The program is a Python function designed to find the number with the highest frequency in a list. You can view the source code at:

<https://github.com/lamtung16/CS567_Project_Software_Testing/blob/main/src_code/mutpy/myCode.py>

I selected this function because it's easy for any programmer to understand and is complex enough for testing purposes. The following subsections demonstrate how to use tstl and mutpy to test the source Python code.

I conducted my experiments using Google Colab due to its Linux environment and convenience for testing, making it a suitable choice for learning purposes.

* 1. **tstl:**
* Install tstl and universalmutator:
  + pip install tstl universalmutator
* In the command line environment:
  + mkdir mutants # Create a directory to store mutant files.
  + tstl test\_myCode.tstl # Generate harness core for testing.
  + mutate myCode.py --mutantDir mutants # Generate mutants for myCode.py and store them in the 'mutants' directory.
  + analyze\_mutants myCode.py "tstl\_rt --timeout 2" --mutantDir mutants --verbose # Analyze all mutants from myCode.py

**2.2 mutpy:**

* Install mutpy:

- pip install mutpy

* In the command line environment:
* mkdir -p report # Create a directory to store .html reports.
* mut.py --target myCode --unit-test test\_myCode --show-mutants --report-html report --timeout-factor 30 --experimental-operators --colored-output # Analyze all mutants from myCode.py using test\_myCode.py

**3. Results**

Screenshots:

|  |  |
| --- | --- |
| mutpy + unittest (.html screenshot) | universalmutator + tstl (CLI screenshot) |
| A screenshot of a test results  Description automatically generated | A close-up of a number  Description automatically generated |

**3.1. Number of mutants:**

During testing, universalmutator generated 53 mutants, surpassing the 23 mutants produced by mutpy. This difference highlights the varying mutation capabilities of the two tools in the context of the given codebase.

**3.2. Quality of mutants:**

Both universalmutator and mutpy produce similar mutants for my source code myCode.py. Most of the mutants are arithmetic operator replacement, assignment operator replacement, conditional operator deletion, one iteration loop, reverse iteration loop, statement deletion, zero iteration loop.

**3.3. Analyzing results:**

When employing unittest with mutpy's mutants, the resulting mutation score reached 82.6%. In contrast, utilizing tstl with universalmutator's mutants yielded a mutation score of 75.5%. These scores provide insights into the effectiveness of the respective testing approaches in identifying and handling mutations introduced by the mutation testing tools.