CS-567

ADVANCED SOFTWARE ASSURANCE

Quality Assurance Report: To-Do List Application

Suresh Perne

6319239

sp2935@nau.edu

Abstract

The testing performed on a To-Do List Application developed as part of a quality assurance (QA) project is detailed in this report. In this, the aim was to see how correct, reliable and robust was the application by doing unit testing, code coverage analysis and mutation testing. Each phase of this report presents what was learned, where areas for improvement can be identified, and recommendations for further refinement.

Source code : https://github.com/Perne007/Suresh_perne_CS567.git

Table of Contents

Introduction

What they wanted was an application to help users manage and track tasks efficiently. Some typical features well supported are the ability to add task, mark task as done, update task details, remove task and by task status or due date.

This report gives valuable insight to the testing strategy used for functionality, performance and maintainability. In order to evaluate how well the test suite covered the application, and to be sure that important aspects of the application were tested, we used a combination traditional unit test, code coverage metrics, and mutation testing.

1. Application Code Overview

The code for the To-Do List application is divided into two primary files:

- **todolist.py**: Contains the business logic for managing tasks.
- **test_todolist.py**: Contains unit tests for the various features of the To-Do List application.

1.1 Core Components of the Application:

- **Task Class**: Individual tasks are represented in this class which is also a to-do list. There are some attributes in each task id, description, due_date and is_completed. Methods for updating task attributes and checking if a task is overdue are included.

```
class Task:
    def __init__(self, id, description, due_date, is_completed=False):
        self.id = id
        self.description = description
        self.due_date = due_date
        self.is_completed = is_completed
```

- **ToDoList Class**: This is a class, which contains a list of Task objects, you can create as many objects as you want inside this class. Users can add tasks, mark them as completed, edit their details and delete tasks. On top of that, it allows you to retrieve tasks getting filtered based on status or due date.

```
class ToDoList:
    def __init__(self):
        self.tasks = []


    def add_task(self, description, due_date):
        task_id = len(self.tasks) + 1
        new_task = Task(task_id, description, due_date)
        self.tasks.append(new_task)
```

2. Testing Strategy

On this perspective, the testing strategy centered on validating all the critical functionalities of the application. This was achieved using the following techniques:

- **Unit Testing:** We tested every method and function individually to ensure that it behaved correctly under a number of conditions.
- **Code Coverage Analysis:** To find how much of the application code was being exercised, we used coverage.py.
- **Mutation Testing:** We ran mutatest to simulate faults in the application and to see if existing tests could detect those mutations.

3. Unit Testing Results

3.1 Test Coverage and Methodology

Unit tests were written for these basic features of the Task and ToDoList classes. I wanted to ensure each feature was functioning correctly and took care of edge cases (such as removing a nonexistent task).Some key tests include:

- **Adding a task**: Verifying that tasks are added correctly and that the task's attributes are properly assigned.
- **Marking a task as completed**: Ensuring that the mark_as_completed method correctly changes the task's status.
- **Removing a task**: Verifying that a task is properly removed from the list.
- **Updating task details**: Ensuring that the description and due date of a task can be updated.
- **Filtering tasks**: Checking if tasks can be filtered based on completion status or due date.

Example test case for adding a task:

```
def test_add_task(self):
    self.todo_list.add_task("New Task", "2025-02-01")
    self.assertEqual(len(self.todo_list.tasks), 4)  # Check if a task was added
    self.assertEqual(self.todo_list.tasks[-1].description, "New Task")  # Check description
    self.assertEqual(self.todo_list.tasks[-1].due_date, "2025-02-01")  # Check due date
```

3.2 Findings and Observations

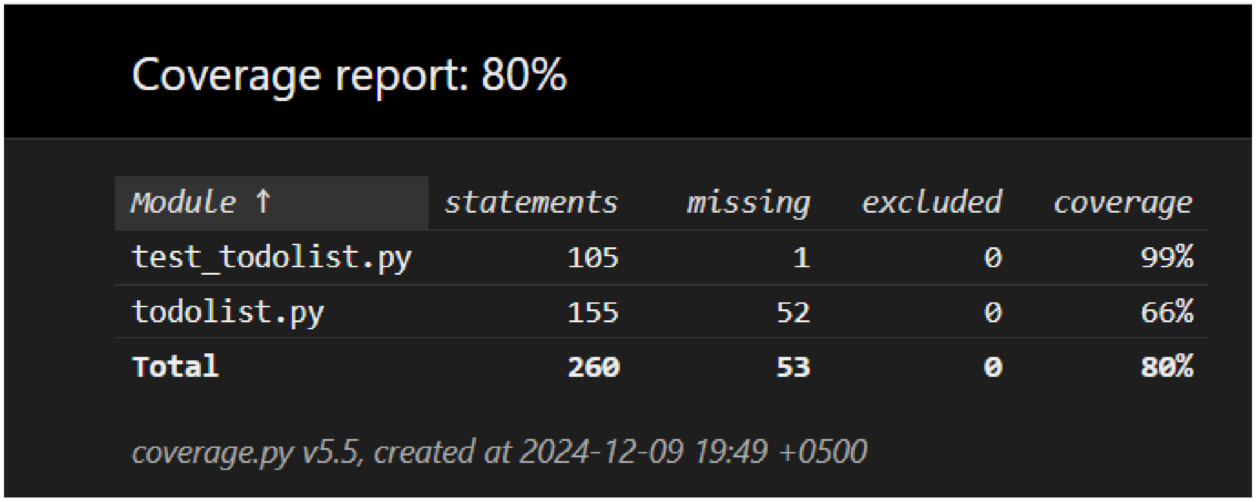
- **Task Addition**: Successfully added tasks, with proper assignment of attributes.

- **Task Removal**: The removal functionality correctly deleted tasks, verified by checks on the task list length.
- **Completion Status**: The mark_as_completed method worked as expected, changing the task's is_completed flag.
- **Edge Case Handling**: Tests were included for edge cases such as removing a non-existent task, updating tasks with invalid data, and handling empty task lists.

4. Code Coverage Analysis

4.1 Overview

It can give an idea of how much code we are testing. The coverage report was generated by usage of coverage.py. Here's the summary:



Coverage report: 80%

Module ↑	statements	missing	excluded	coverage
test_todolist.py	105	1	0	99%
todolist.py	155	52	0	66%
Total	**260**	**53**	**0**	**80%**

coverage.py v5.5, created at 2024-12-09 19:49 +0500

4.2 Key Insights:

- The **test file (test_todolist.py)** has **99% coverage**, indicating that almost all test scenarios are being properly exercised.
- However, the **source file (todolist.py)** has **66% coverage**, suggesting that some areas of the core logic, particularly more complex branches and edge cases, are not fully tested.

4.3 Recommendations for Improved Coverage:

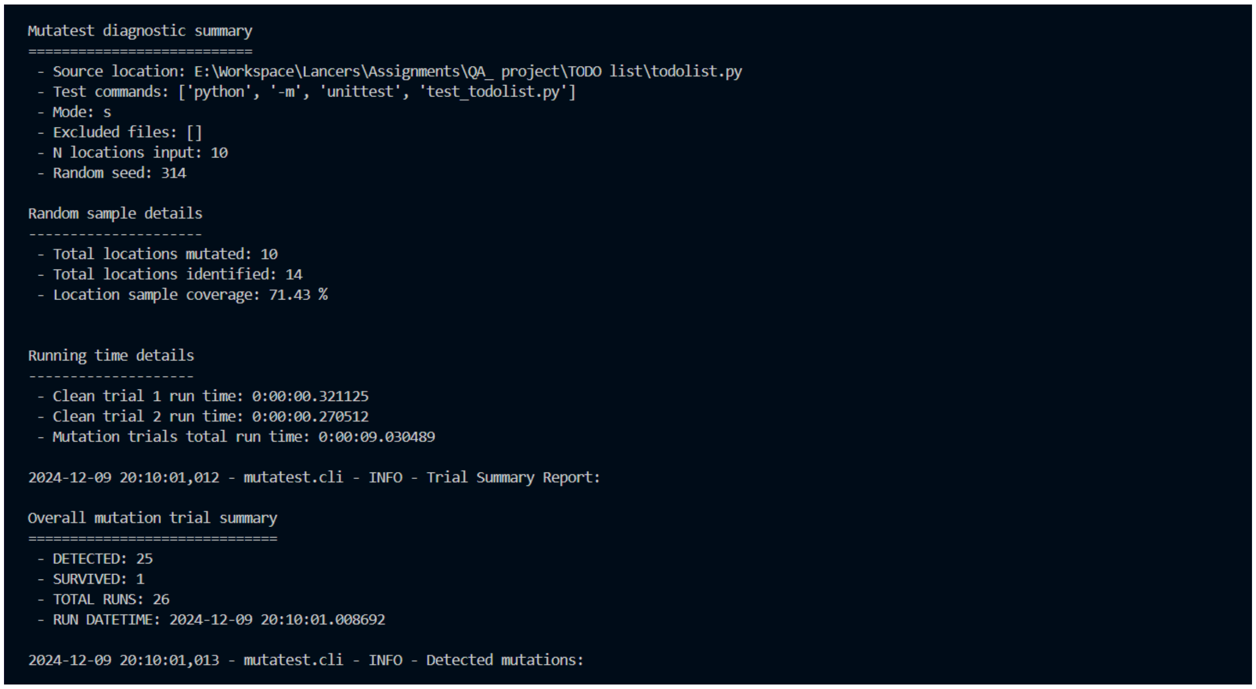
- Having already examined code that's not tested, there are parts which should have their own tests: complex conditional branches or edge cases like invalid date formats or extreme task statuses.
- Refactor large methods or functions to become testable, so it's easier to break them down into smaller, single unit tests.

5. Mutation Testing Results

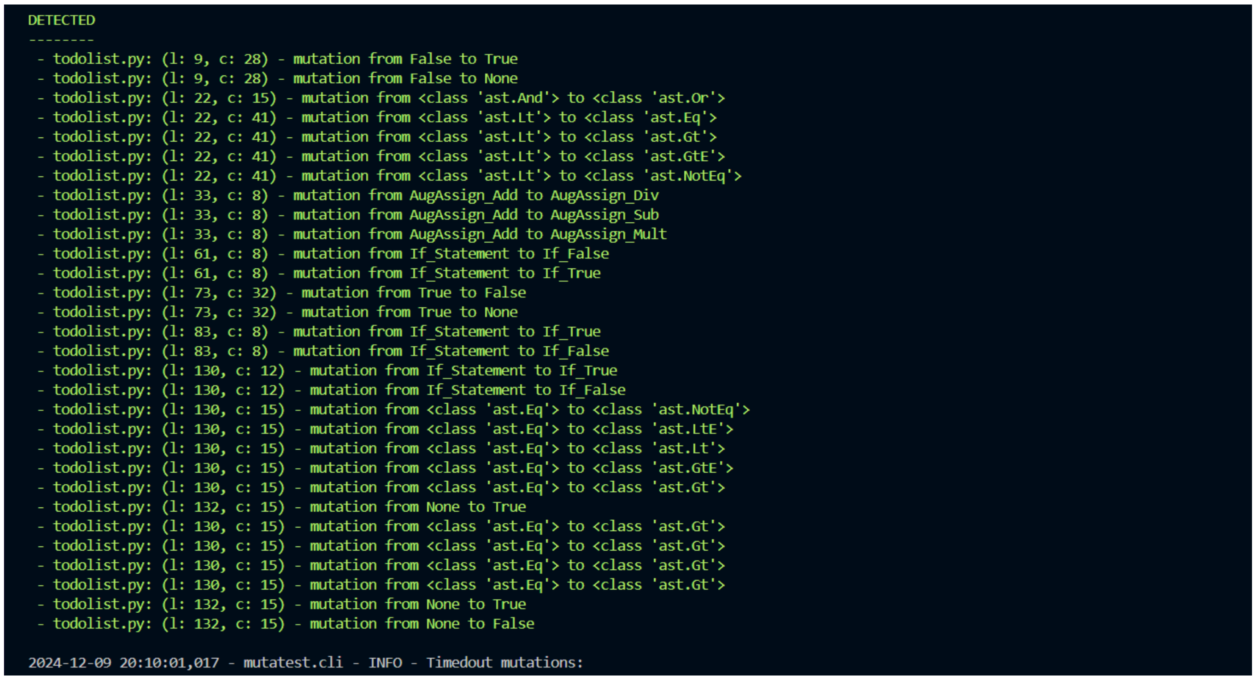
5.1 Overview

The effectiveness of the unit tests in detecting subtle faults in the application code was measured with mutation testing. We ran the small changes (mutations) in the source code through the mutatest tool to see if existing tests could find these mutations.

5.2 Mutation Test Summary:



```
Mutatest diagnostic summary
===========================
 - Source location: E:\Workspace\Lancers\Assignments\QA_ project\TODO list\todolist.py
 - Test commands: ['python', '-m', 'unittest', 'test_todolist.py']
 - Mode: s
 - Excluded files: []
 - N locations input: 10
 - Random seed: 314

Random sample details
---------------------
 - Total locations mutated: 10
 - Total locations identified: 14
 - Location sample coverage: 71.43 %


Running time details
--------------------
 - Clean trial 1 run time: 0:00:00.321125
 - Clean trial 2 run time: 0:00:00.270512
 - Mutation trials total run time: 0:00:09.030489

2024-12-09 20:10:01,012 - mutatest.cli - INFO - Trial Summary Report:

Overall mutation trial summary
==============================
 - DETECTED: 25
 - SURVIVED: 1
 - TOTAL RUNS: 26
 - RUN DATETIME: 2024-12-09 20:10:01.008692

2024-12-09 20:10:01,013 - mutatest.cli - INFO - Detected mutations:
```



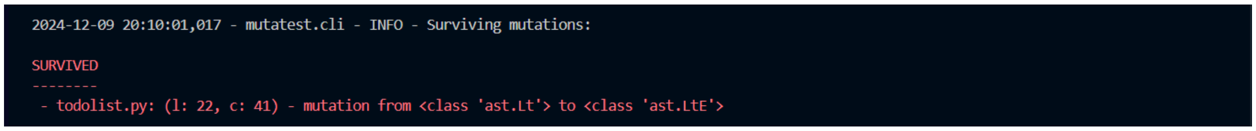
```
DETECTED
--------
 - todolist.py: (l: 9, c: 28) - mutation from False to True
 - todolist.py: (l: 9, c: 28) - mutation from False to None
 - todolist.py: (l: 22, c: 15) - mutation from <class 'ast.And'> to <class 'ast.Or'>
 - todolist.py: (l: 22, c: 41) - mutation from <class 'ast.Lt'> to <class 'ast.Eq'>
 - todolist.py: (l: 22, c: 41) - mutation from <class 'ast.Lt'> to <class 'ast.Gt'>
 - todolist.py: (l: 22, c: 41) - mutation from <class 'ast.Lt'> to <class 'ast.GtE'>
 - todolist.py: (l: 22, c: 41) - mutation from <class 'ast.Lt'> to <class 'ast.NotEq'>
 - todolist.py: (l: 33, c: 8) - mutation from AugAssign_Add to AugAssign_Div
 - todolist.py: (l: 33, c: 8) - mutation from AugAssign_Add to AugAssign_Sub
 - todolist.py: (l: 33, c: 8) - mutation from AugAssign_Add to AugAssign_Mult
 - todolist.py: (l: 61, c: 8) - mutation from If_Statement to If_False
 - todolist.py: (l: 61, c: 8) - mutation from If_Statement to If_True
 - todolist.py: (l: 73, c: 32) - mutation from True to False
 - todolist.py: (l: 73, c: 32) - mutation from True to None
 - todolist.py: (l: 83, c: 8) - mutation from If_Statement to If_True
 - todolist.py: (l: 83, c: 8) - mutation from If_Statement to If_False
 - todolist.py: (l: 130, c: 12) - mutation from If_Statement to If_True
 - todolist.py: (l: 130, c: 12) - mutation from If_Statement to If_False
 - todolist.py: (l: 130, c: 15) - mutation from <class 'ast.Eq'> to <class 'ast.NotEq'>
 - todolist.py: (l: 130, c: 15) - mutation from <class 'ast.Eq'> to <class 'ast.LtE'>
 - todolist.py: (l: 130, c: 15) - mutation from <class 'ast.Eq'> to <class 'ast.Lt'>
 - todolist.py: (l: 130, c: 15) - mutation from <class 'ast.Eq'> to <class 'ast.GtE'>
 - todolist.py: (l: 130, c: 15) - mutation from <class 'ast.Eq'> to <class 'ast.Gt'>
 - todolist.py: (l: 132, c: 15) - mutation from None to True
 - todolist.py: (l: 130, c: 15) - mutation from <class 'ast.Eq'> to <class 'ast.Gt'>
 - todolist.py: (l: 130, c: 15) - mutation from <class 'ast.Eq'> to <class 'ast.Gt'>
 - todolist.py: (l: 130, c: 15) - mutation from <class 'ast.Eq'> to <class 'ast.Gt'>
 - todolist.py: (l: 130, c: 15) - mutation from <class 'ast.Eq'> to <class 'ast.Gt'>
 - todolist.py: (l: 132, c: 15) - mutation from None to True
 - todolist.py: (l: 132, c: 15) - mutation from None to False

2024-12-09 20:10:01,017 - mutatest.cli - INFO - Timedout mutations:
```

5.3 Key Findings:

- **Detected Mutations**: **25 out of 26** mutations were successfully identified, demonstrating that the tests can detect most types of faults in the code.
- **Surviving Mutation**: One mutation survived, which involved changing a comparison operator from < to <=. This suggests that some areas, particularly comparisons involving task due dates or statuses, are not adequately tested.

5.4 Surviving Mutation Analysis:

- Mutations that survived indicate that tests may not cover boundary cases around comparison operators.. The detection of such mutations may be improved by more targeted tests for these scenarios.



```
2024-12-09 20:10:01,017 - mutatest.cli - INFO - Surviving mutations:

SURVIVED
--------
 - todolist.py: (l: 22, c: 41) - mutation from <class 'ast.Lt'> to <class 'ast.LtE'>
```

6. Recommendations for Improvement

Based on the findings from unit testing, code coverage analysis, and mutation testing, the following improvements are recommended:

1. **Expand Test Coverage**:
   - Write additional tests to cover edge cases, particularly for comparisons and date validation.
   - Ensure that every method and branch is covered by at least one unit test.
2. **Enhance Tests for Comparison Logic**:
   - Create more focused tests around task due dates and their comparison operators (e.g., <=, >=, <, >).
   - Ensure that boundary conditions (such as dates that are on the edge of being overdue) are explicitly tested.
3. **Refactor Code for Testability**:
   - Consider breaking down larger functions into smaller, more testable components to ensure that each piece of logic can be properly verified.
4. **Automate Testing with Continuous Integration**:
   - Implement continuous integration (CI) practices to run tests automatically when code changes are made. This will help maintain high-quality standards over time.

7.Conclusion

The To-Do List Application has been proven with unit tests, code coverage and mutation testing. Results show that the application is robust on the whole (though there are improvements to be made to test coverage, in particular around complex conditional branches and edge cases).

After following these recommendations, the application will reach even higher levels of quality and reliability to create a nice user experience.

References

1. **Python Unit Testing Documentation**
   Python Software Foundation. (n.d.). unittest — Unit testing framework. Python Docs.
   https://docs.python.org/3/library/unittest.html
2. **coverage.py Documentation**
   Haller, M. (2020). coverage.py — Code coverage measurement for Python.
   https://coverage.readthedocs.io/en/coverage-5.5/
3. **Mutation Testing Resources**
   The Mutation Testing Alliance. (2022). Mutation Testing: A Comprehensive Guide.
   https://mutation-testing.org/
4. **Clean Code: A Handbook of Agile Software Craftsmanship** by Robert C. Martin (2008). This book provides best practices for writing clean, testable code.