Software Unit Testing Report

PRT582

Software Engineering: Process & Tools

PREPARED FOR

Charles Yeo

CHARLES DARWIN UNIVERSITY

PREPARED BY

TAUHIDUL ISLAM ATOUL - S373797

MASTER OF INFORMATION TECHNOLOGY (SOFTWARE ENGINEERING)

CHARLES DARWIN UNIVERSITY

DATE OF SUBMISSION:

06-September-2024

Scrabble Game Project Report

Introduction

This project focused on creating a Scrabble score calculator game in Python using Test-Driven Development (TDD). The game calculates Scrabble scores for words based on predefined letter values, validates the input, and checks if the word is valid using an online dictionary API.

We chose Python for its simplicity, readability, and robust testing support. Python's `unittest` module is built-in, making applying TDD easy and ensuring the program works as expected.

Process

TDD and Automated Unit Testing

In this project, we followed TDD by writing test cases first, implementing the corresponding code, and refining the implementation until the tests passed. We used Python's `unittest` framework to automate tests for each feature.

1. **Scrabble Score Calculation:** We wrote tests to ensure the program calculates scores correctly for both lowercase and uppercase letters.
2. **Input Validation:** Tests were written to handle invalid inputs, ensuring only alphabetic characters are accepted. If the input is invalid, the program raises an error.
3. **Random Word Length with Timer:** We generated a random word length, tested that input is provided within 15 seconds, and validated whether the user's input meets the required length.
4. **Dictionary Validation:** We used an API to check whether the entered word exists in a real dictionary and wrote tests to verify this functionality.
5. **Game Loop:** The game runs for 10 rounds, displaying the total score. The game ensures valid input in every round, with the option to quit anytime.

Lessons Learned

1. **The Value of TDD:** Test-Driven Development (TDD) ensured we approached the problem methodically. By writing tests first, we were able to clearly define the program's requirements and avoid common mistakes. It helped make the code more reliable and maintainable.
2. **Importance of Input Validation:** Validating user input early in the process (e.g., checking for non-alphabetic characters, word length) made the game more robust. It highlighted how important it is to anticipate user behavior and handle incorrect inputs gracefully.

3. **Automated Testing Saves Time:** Using Python's `unittest` framework allowed us to run automated tests quickly and repeatedly. This made it easier to identify and fix bugs, especially when refactoring or adding new features.
4. **Using External APIs:** Incorporating an online dictionary API for word validation added a level of complexity but made the game more realistic and educational. It also taught us how to handle external dependencies (e.g., network issues or API response failures) in a Python program.
5. **Interactive Features:** Adding a timer and random word length added an interesting layer of challenge to the game. It showed how user experience can be enhanced through thoughtful feature design.

How It Can Be Improved

1. **Enhanced Dictionary Validation:** The current implementation uses an external API to validate words, which depends on a stable internet connection. For offline use, integrating a local dictionary file could improve performance and reduce reliance on external services.
2. **Better Error Handling:** While basic input validation was implemented, the program could benefit from more sophisticated error handling, such as providing clearer feedback for invalid input, handling edge cases (e.g., very long words), or offering retry options.
3. **Improve Game Features:** Adding more features to the game, such as allowing the user to choose the number of rounds, introducing difficulty levels, or even adding multiplayer options, could make it more engaging.
4. **Performance Optimization:** For very large word lists or frequent dictionary checks, performance may degrade. Implementing caching mechanisms for word validation or optimizing score calculations could improve speed and efficiency.
5. **User Interface (UI):** Currently, the game runs in the command line, but developing a graphical user interface (GUI) using libraries like Tkinter or PyQt would make the game more visually appealing and user-friendly.

Conclusion

This project highlighted the benefits of TDD. Writing tests first ensured that we built a reliable program step-by-step, making debugging easier and improving code quality. The use of automated testing with `unittest` provided confidence during development, especially when refactoring code.

You can find the complete source code visit the github repository by clicking here