

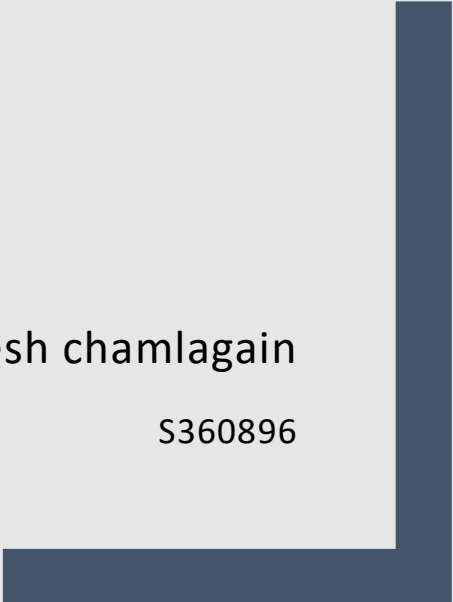


PRT582 Software Engineering: Process and Tools

# SOFTWARE UNIT TESTING REPORT

yukesh chamlagain

S360896



## Introduction

Ensuring code's dependability, correctness, and functionality is critical in software development. The "Guess the Number" game project was undertaken to develop a functioning and engaging game while adhering to stringent testing procedures. This paper summarises the project's goals, needs, and the use of automated unit testing techniques to achieve excellent code quality.

## Objectives and Requirements

The project's primary goal was to create and build a "Guess the Number" game in which participants had limited attempts to predict a randomly generated 4-digit number. The primary functions of the game included generating random numbers, evaluating player guesses, and delivering relevant clues based on comparing the guessed number and the target number.

To achieve these goals, the project used the following guidelines:

- **Interactive Gameplay:** Create a fully working "Guess the Number" game in which participants can guess the correct number interactively.
- **Automated Testing:** Use automated unit tests to validate the validity and accuracy of the fundamental game functionalities, guaranteeing that the game works as intended and meets the requirements.
- **Code Quality:** Maintain good code quality by adhering to established coding standards, creating a beautiful and modular code design, and using relevant comments to improve code readability.

## Automated Unit Testing Tool

The project used the unittest framework, a built-in testing framework in Python, to achieve reliable and methodical testing. unittest provides a robust and disciplined way to develop and run tests, allowing flaws to be identified early in the development process. The framework allowed for the design of test cases that tested individual pieces of code, ensuring that each function functioned properly and delivered the intended outcomes.

The following sections of this report detail the project's implementation, the technique used for automated unit testing, the code design, and the testing results. The project aimed to

create a functioning and well-tested "Guess the Number" game that adhered to best practices in software development by merging these features.

### **Process: Test-Driven Development and Automated Unit Testing**

#### **Test Driven Development (TDD)**

The project's development strategy was founded on Test-Driven Development (TDD), an iterative approach to software development that emphasises establishing tests before producing code. By methodically verifying its functionality against predicted outcomes, this methodology encouraged the development of well-structured, well-tested code.

During the project, the TDD cycle consisted of three stages:

- **Write a Test:** A related test case was designed using the unittest framework before implementing any new functionality or making changes. These test cases were written to illustrate certain circumstances and ensure that each function generated the expected outcomes.
- **Fail the Test:** With the test cases in place, the code was run against the tests until the test passed.
- **Write the Code:** The appropriate code was then developed or adjusted to meet the requirements of the test cases. This phase provides the desired functionality and ensures the code passes the written tests.

The development process was driven by a thorough validation procedure that guaranteed new code complied with the project's specifications and requirements by repeating this cycle.

#### **Utilisation of the Automated Unit Testing Tool**

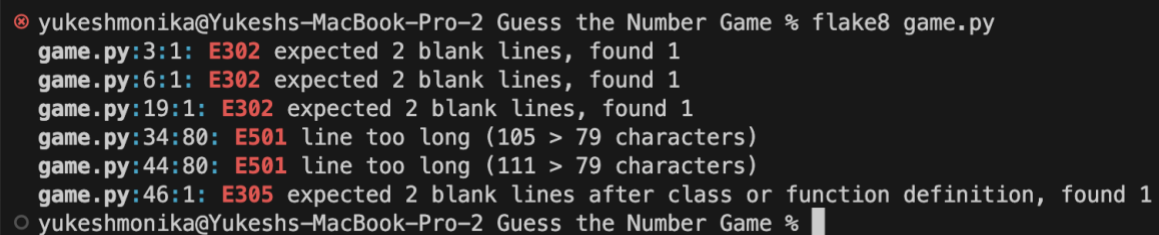
The unittest framework was used to aid with the development of TDD and the execution of rigorous testing. This Python built-in testing framework enabled the production of organised and structured test suites, each including many test cases aimed at specific features—the framework's capabilities allowed for the automation of test execution and the reporting of test results.

## Code Quality and the Linting Tools (Pylint and Flake8)

In addition to automated testing, the project recognised the importance of code quality by using linting tools such as Pylint and Flake8. Pylint provides thorough code analysis, finding potential problems, conforming to coding standards, and encouraging consistent code style. Flake8 prioritises code formatting and adherence to PEP 8 rules to keep the codebase legible and maintainable.

### Flake8

Below are the screenshots of tests run using flake8,

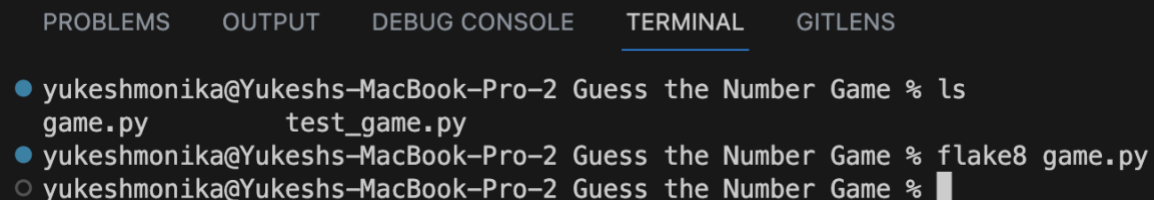


```
yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game % flake8 game.py
game.py:3:1: E302 expected 2 blank lines, found 1
game.py:6:1: E302 expected 2 blank lines, found 1
game.py:19:1: E302 expected 2 blank lines, found 1
game.py:34:80: E501 line too long (105 > 79 characters)
game.py:44:80: E501 line too long (111 > 79 characters)
game.py:46:1: E305 expected 2 blank lines after class or function definition, found 1
yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game %
```

After the test, the following errors were encountered.

- The error E302 indicates that there should be two blank lines before the 'def' statement.
- The error E501 indicates that lines 34 and 44 exceed the recommended maximum line length of 79 characters according to the PEP 8 coding style guidelines.
- The error E305 indicates that there should be two blank lines after the function or class definition.

After resolving all the issues and running the flake8 test, the result looked like the screenshot below,



```
yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game % ls
game.py      test_game.py
yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game % flake8 game.py
yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game %
```

Similarly, several flake8 tests were run on the 'test\_game.py' file to get the accurate code, which is demonstrated by the screenshot below,

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  GITLENS

tets_game.py:0:1: E902 FileNotFoundError: [Errno 2] No such file or directory: 'tets_game.py'
yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game % flake8 test_game.py
test_game.py:4:1: E302 expected 2 blank lines, found 1
test_game.py:17:1: E305 expected 2 blank lines after class or function definition, found 1
test_game.py:19:1: W293 blank line contains whitespace
yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game % flake8 test_game.py
test_game.py:20:20: W292 no newline at end of file
yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game % flake8 test_game.py
test_game.py:21:1: W293 blank line contains whitespace
test_game.py:21:5: W292 no newline at end of file
yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game % flake8 test_game.py
test_game.py:21:1: W293 blank line contains whitespace
test_game.py:21:5: W292 no newline at end of file
yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game % flake8 test_game.py
yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game % █

```

- The error E302 indicates that there should be two blank lines before the 'def' statement.
- The error E305 indicates that there should be two blank lines after the function or class definition.
- The error W293 indicates that there is a whitespace in the blank line.
- The error 292 indicates no newline at the end of the file.

All the issues were figured out using flake8 and resolved, which is demonstrated by the last line of the above screenshot.

## Pylint

Pylint was also used to check the errors. Below are the screenshots of the tests and how I resolved the issues,

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  GITLENS

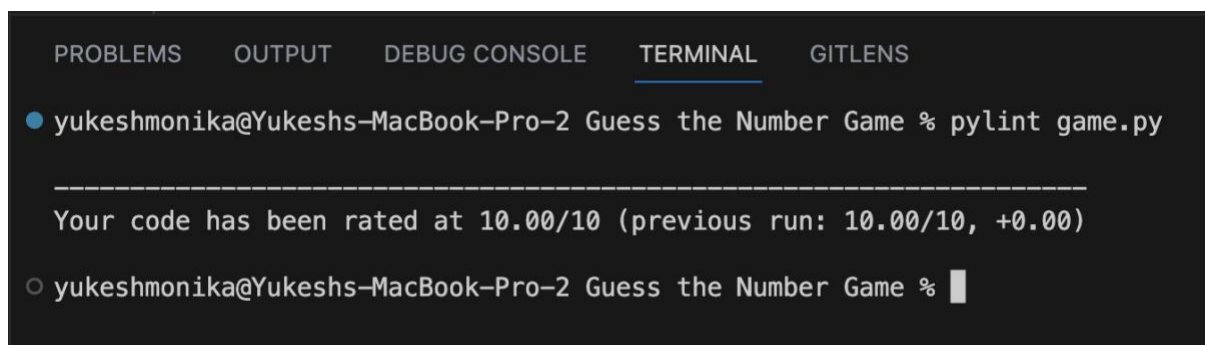
yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game % pylint game.py
***** Module game
game.py:30:0: C0325: Unnecessary parens after 'not' keyword (superfluous-parens)
game.py:1:0: C0114: Missing module docstring (missing-module-docstring)
game.py:4:0: C0116: Missing function or method docstring (missing-function-docstring)
game.py:8:0: C0116: Missing function or method docstring (missing-function-docstring)
game.py:22:0: C0116: Missing function or method docstring (missing-function-docstring)
game.py:36:12: R1723: Unnecessary "else" after "break", remove the "else" and de-indent the code inside it (no-else-break)

-----
Your code has been rated at 8.24/10 (previous run: 7.31/10, +0.93)
yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game % █

```

- The error C0325 indicates unnecessary parentheses around the 'not' keyword.
- The error C0114 suggest adding a docstring at the beginning of the module to explain its purpose and functionality.
- The error C0116 indicates that the function or the method docstrings are missing.
- The error R1723 suggest removing the 'else' block following the 'break' statement since code after 'break' will never be executed.

After resolving all the issues, the Pylint test result was obtained, as shown in the screenshot below,



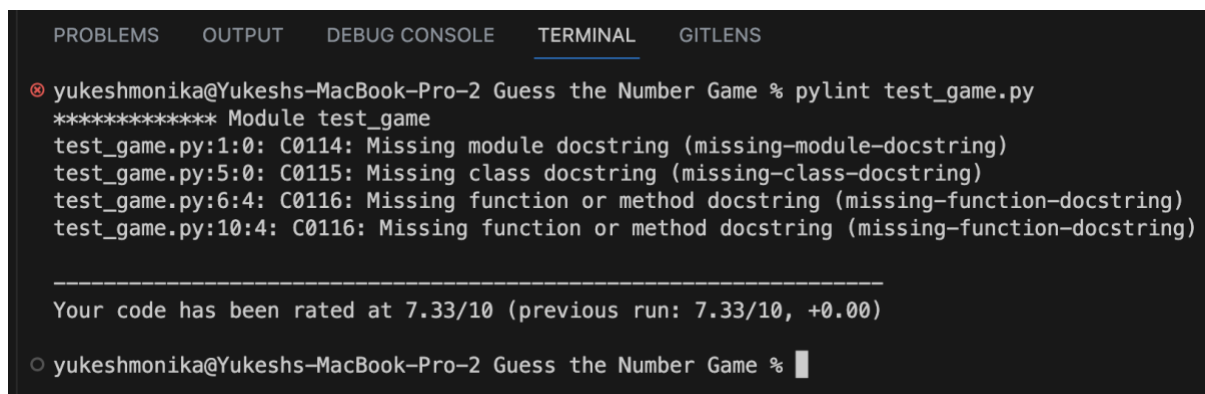
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  GITLENS

● yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game % pylint game.py

-----
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)

○ yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game %
```

Similarly, several Pylint tests were run on the 'test\_game.py' file to get the accurate code, which is demonstrated by the screenshot below,



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  GITLENS

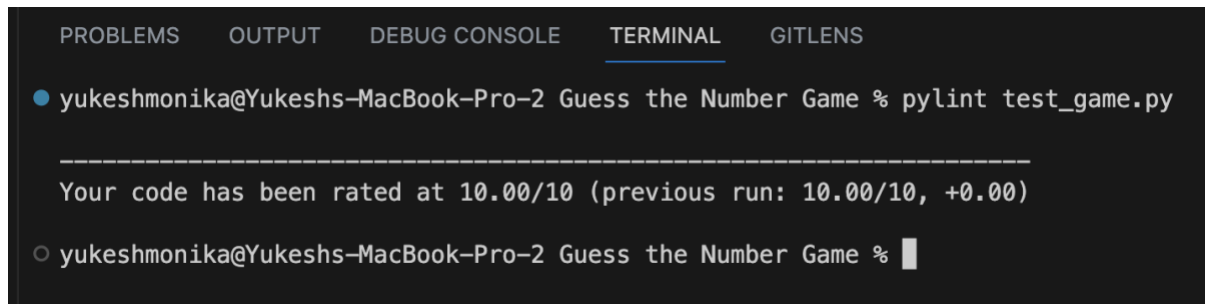
⊗ yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game % pylint test_game.py
***** Module test_game
test_game.py:1:0: C0114: Missing module docstring (missing-module-docstring)
test_game.py:5:0: C0115: Missing class docstring (missing-class-docstring)
test_game.py:6:4: C0116: Missing function or method docstring (missing-function-docstring)
test_game.py:10:4: C0116: Missing function or method docstring (missing-function-docstring)

-----
Your code has been rated at 7.33/10 (previous run: 7.33/10, +0.00)

○ yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game %
```

- The error C0114 indicates no docstring at the beginning of the module (file) explaining its purpose and functionality.
- The error C0115 indicates that there's no docstring for the test class.
- The error C0116 indicates that there are no docstrings for the test methods.

After resolving all the issues, the Pylint test result was obtained, as shown in the screenshot below,



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  GITLENS

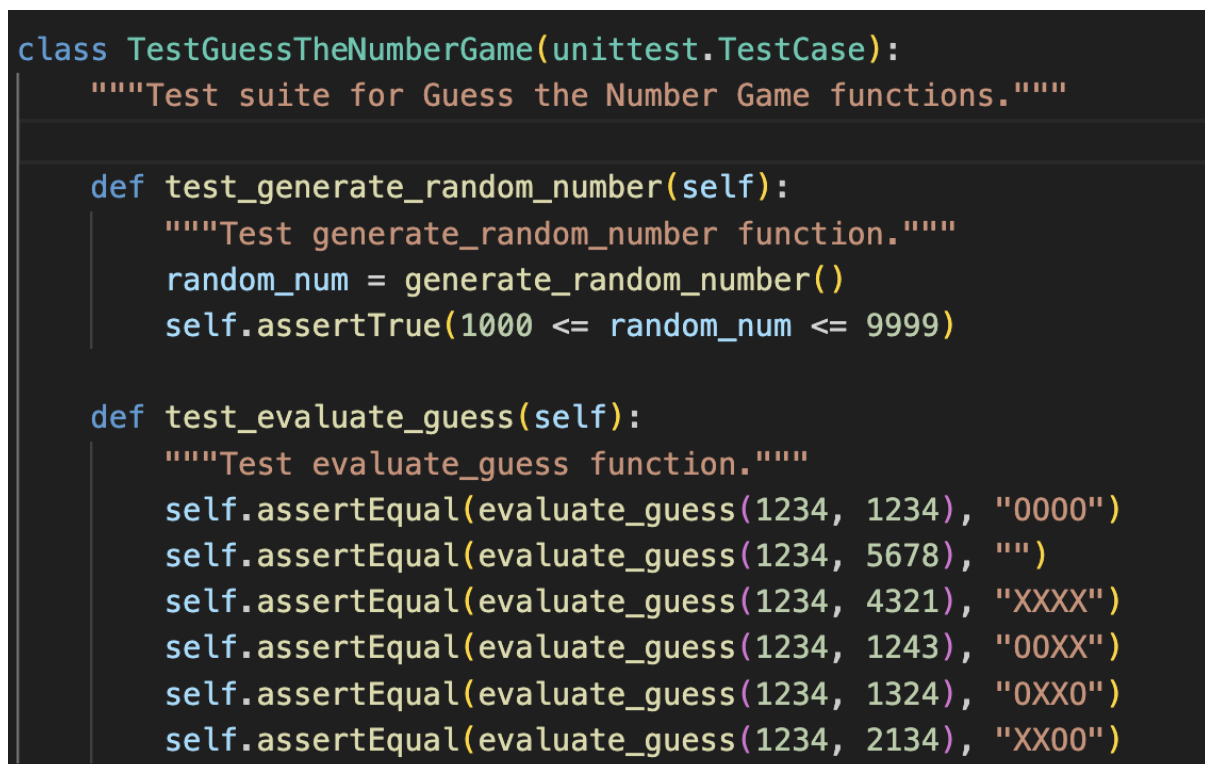
● yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game % pylint test_game.py

-----
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)

○ yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game %
```

### Screenshots of Test cases and Execution

Below is the screenshot of **Generate Random Number Test** and **Evaluate Guess Test**



```
class TestGuessTheNumberGame(unittest.TestCase):
    """Test suite for Guess the Number Game functions."""

    def test_generate_random_number(self):
        """Test generate_random_number function."""
        random_num = generate_random_number()
        self.assertTrue(1000 <= random_num <= 9999)

    def test_evaluate_guess(self):
        """Test evaluate_guess function."""
        self.assertEqual(evaluate_guess(1234, 1234), "0000")
        self.assertEqual(evaluate_guess(1234, 5678), "")
        self.assertEqual(evaluate_guess(1234, 4321), "XXXX")
        self.assertEqual(evaluate_guess(1234, 1243), "00XX")
        self.assertEqual(evaluate_guess(1234, 1324), "0XX0")
        self.assertEqual(evaluate_guess(1234, 2134), "XX00")
```

The **'test\_generate\_random\_number()'** case assesses the **'generate\_random\_number()'** function. This automated test guarantees the random number's conformance to the specified range, preserving the game's core randomness.

The **'test\_evaluate\_guess()'** case elucidates the efficacy of the **'evaluate\_guess()'** function. By comparing anticipated hints with actual outputs, this automated testing process guarantees the precision of hint generation.

Below is the screenshot of unittest results obtained after doing the test in both files 'test\_game.py' and 'game.py',

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  GITLENS

● yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game % ls
game.py      test_game.py
● yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game % python -m unittest test_game
..
-----
Ran 2 tests in 0.000s

OK
○ yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game %
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  GITLENS

● yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game % python -m unittest game
-----
Ran 0 tests in 0.000s

OK
○ yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game %
```

Below is the final output of the game,

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  GITLENS

Enter your guess (4-digit number): 1222
Hint: 0
Enter your guess (4-digit number): 1333
Hint: 0
Enter your guess (4-digit number): 1444
Hint: 0XX0
Enter your guess (4-digit number): 1554
Hint: 00
Enter your guess (4-digit number): 1664
Hint: 0X00
Enter your guess (4-digit number): 1764
Hint: 000
Enter your guess (4-digit number): 1864
Congratulations! You guessed the number 1864 in 8 attempts.
○ yukeshmonika@Yukeshs-MacBook-Pro-2 Guess the Number Game %
```



## Conclusion

The "Guess the Number" game project experience has provided invaluable insights into software development, automated testing, and code quality complexities. The project produced positive results while highlighting areas that needed improvement by following to Test-Driven Development (TDD) principles and utilising automated unit testing tools.

Several major lessons developed along the course of the project:

**Rigours Testing Improves Reliability:** TDD and automated testing has highlighted the relevance of rigours testing. Early detection of flaws allowed for fast corrective measures, increasing the codebase's reliability.

**Linting Tools Encourage Consistency:** The integration of Pylint and Flake8 highlighted the importance of consistent coding practices. Enforcing coding standards and stylistic rules improved code readability and collaboration.

## Improvement Opportunities

While the project met its primary goals, a few elements stand out for future improvement:

**User Input Validation:** Improving user input validation can help to avoid unexpected errors and improve the user experience. Incorporating robust validation techniques can help to ensure that players provide accurate guesses.

**Interactive User Interface:** Enhancing the user experience with an interactive and visually appealing interface could make the game more exciting and accessible.

## Prospects for the Future

The "Guess the Number" project can be expanded and improved in the following ways:

**Multiplayer Mode:** A multiplayer feature in which players compete to guess the number in the fewest attempts could add a competitive element to the game.

**Difficulty Levels:** Implementing several difficulty levels with distinct goal number ranges could cater to many players.

**Access to the GitHub Repository**

The whole project report and source code may be found in the following GitHub repository:

<https://github.com/YukeshChamlagain98/software-Unit-Testing-Report>