HW7 - Software Project Management

Software Testing

Overview:

The analysis of the software testing approach for this project encompasses a comprehensive strategy that includes both unit and acceptance tests for two distinct features: Feature A (SprintVelocityCalculator) and Feature B (TeamCapacity). This testing strategy is meticulously designed to ensure thorough coverage, evaluating the functionalities across both happy and unhappy paths to ascertain that all possible method execution paths are effectively examined.

UNIT TEST:

The provided unit tests for both Feature A (SprintVelocityCalculator) and Feature B (TeamCapacity) have been designed to comprehensively test the functionalities by covering both happy and unhappy paths, ensuring that all possible paths through the methods are evaluated.

Feature A: Testing SprintVelocityCalculator

Happy Path:

Method Tested: calculate_average_velocity

Test Case: test_average_velocity_with_initial_points

Scenario: The test creates an instance of SprintVelocityCalculator with an initial list of sprint points [30, 40, 50, 45] and asserts that the calculate_average_velocity method returns the correct average velocity (41.25). This represents the happy path where the method processes valid input data and calculates the expected result.

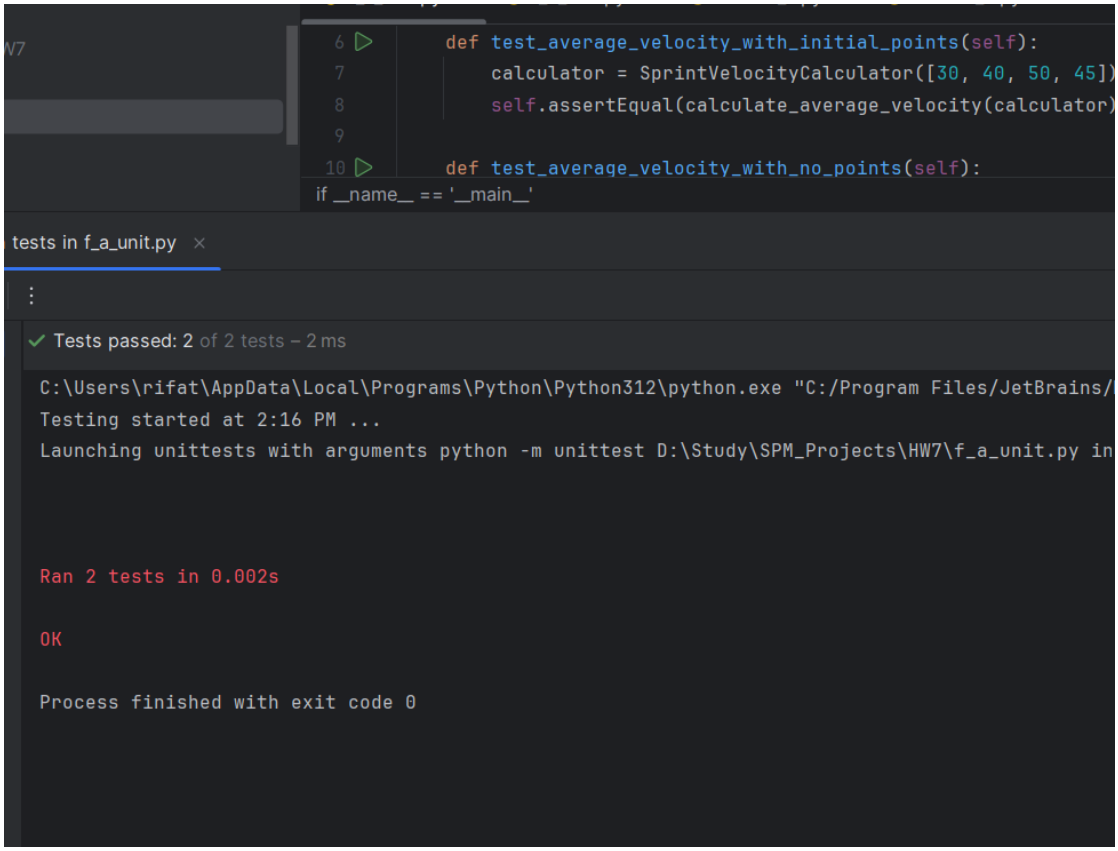
Unhappy Path:

Method Tested: calculate_average_velocity

Test Case: test_average_velocity_with_no_points

Scenario: The test initializes SprintVelocityCalculator with an empty list of sprint points and asserts that calculate_average_velocity correctly returns 0, handling the case where there are no sprint points gracefully. This represents the unhappy path, testing the method's ability to handle edge cases or less-than-ideal input conditions.

Feature_A unit test:



Feature B: Testing TeamCapacity and Related Functionalities

Happy Path:

Method Tested: calculate_effort_minutes

Test Case: test_effort_calculation_with_team_members

Scenario: The setUp method initializes a TeamCapacity instance for a 10-day sprint and adds two team members with specified available minutes per day, days off, and minutes

committed to ceremonies. The test_effort_calculation_with_team_members case then calculates the total and per person effort using calculate_effort_minutes, comparing the results against manually calculated expected values. This test validates that the method accurately calculates effort based on realistic team member data, representing the happy path where data is complete and valid.

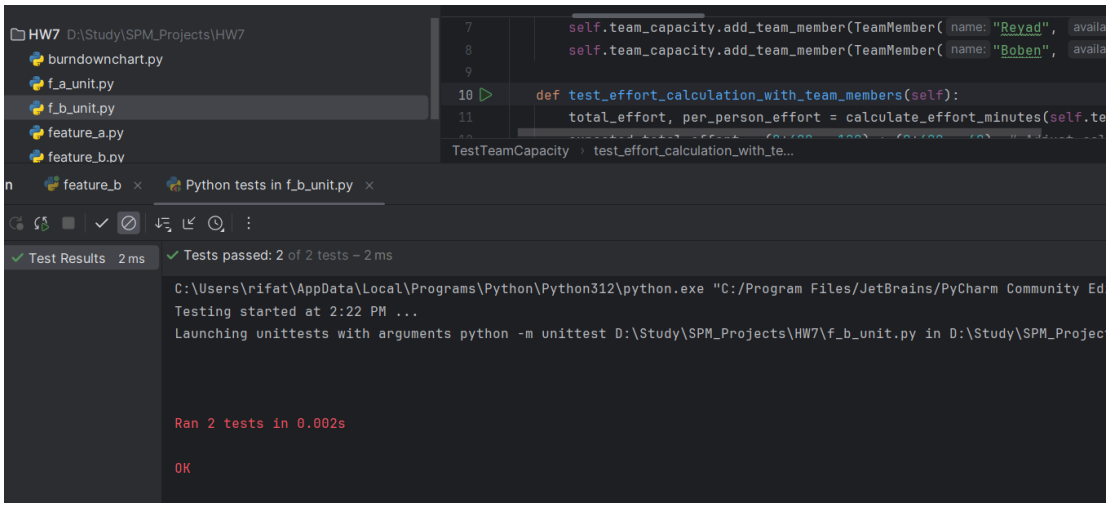
Unhappy Path:

Method Tested: calculate_effort_minutes

Test Case: test_effort_calculation_without_team_members

Scenario: This test initializes a TeamCapacity instance without adding any team members, simulating an empty team scenario. It then asserts that calculate_effort_minutes returns 0 for both total effort and per person effort, testing the method's handling of situations where there is no data to process. This represents the unhappy path, ensuring the system behaves correctly in edge cases or scenarios with missing input.

Feature_b_unit test:



Summary of Approach:

These tests are structured to ensure both main functionalities (calculate_average_velocity for Feature A and calculate_effort_minutes for Feature B) work correctly under both ideal conditions (happy paths) and less-than-ideal or edge conditions (unhappy paths). By doing so, they validate that the software is robust, handles errors gracefully, and meets the

functional requirements across a range of scenarios. This comprehensive testing approach is essential for building confidence in the software's reliability and correctness.

Acceptance Test:

The acceptance tests for both Feature A and Feature B have been thoughtfully designed to meet the requirement of exercising more than a single method while addressing both happy and unhappy path scenarios.

Feature A Acceptance Test Methods:

Scenario 1 (Happy Path):

SprintVelocityCalculator([30, 40, 50, 45]): Constructor method to initialize the calculator with a list of sprint points.

calculate_average_velocity(calculator): Method to calculate the average velocity based on the provided sprint points.

display_velocity(calculator): Method to display the calculated average velocity.

Scenario 2 (Unhappy Path):

SprintVelocityCalculator([]): Constructor method to initialize the calculator with an empty list, simulating no sprint points.

calculate_average_velocity(calculator): Method to attempt calculating the average velocity with no points, expecting to handle this gracefully.

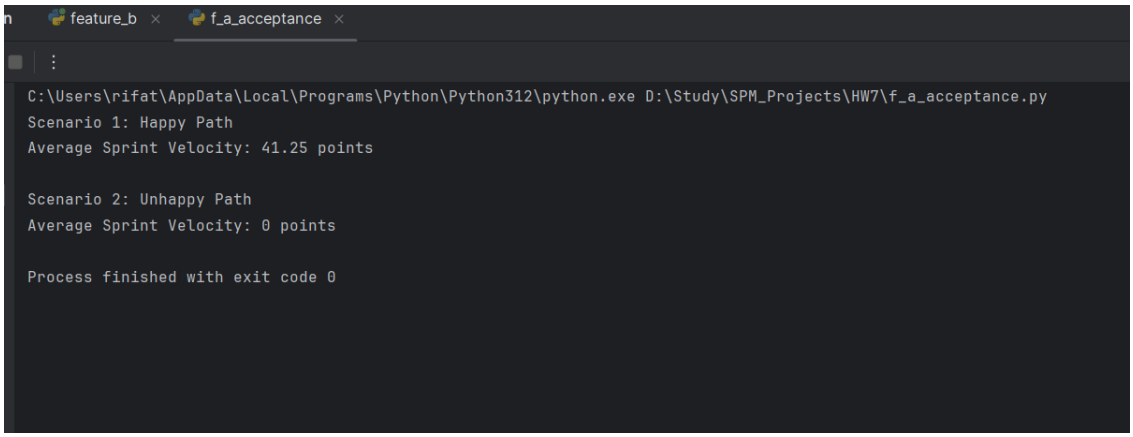
display_velocity(calculator): Method to display the outcome, expected to show an average velocity of 0.

Analysis of Scenarios:

Happy Path (Scenario 1): Initializes the calculator with a list of sprint points and validates that the calculated and displayed average velocity is correct, demonstrating the integration and proper function of the constructor, calculation, and display methods.

Unhappy Path (Scenario 2): Initializes the calculator with an empty list, showcasing the system's ability to handle cases with no data and still function correctly by calculating and displaying an average velocity of 0.

Feature_a_acceptance_test:



Feature B Acceptance Test Methods:

Scenario 1 (Happy Path):

TeamCapacity(10): Constructor method to create a team capacity instance for a 10-day sprint.

add_team_member(TeamMember("Alex", 480, 1, 120)) and subsequent calls: Method to add team members with their availability and commitments.

calculate_effort_minutes(team_capacity): Method to calculate total and per person effort-hours based on team member details.

display_team_capacity(team_capacity): Method to display the calculated effort-hours for verification.

Scenario 2 (Unhappy Path):

TeamCapacity(10): Constructor method to create a team capacity instance without adding any team members, simulating an empty team.

calculate_effort_minutes(team_capacity_empty): Method to calculate effort-hours with no team members, expecting to handle this gracefully.

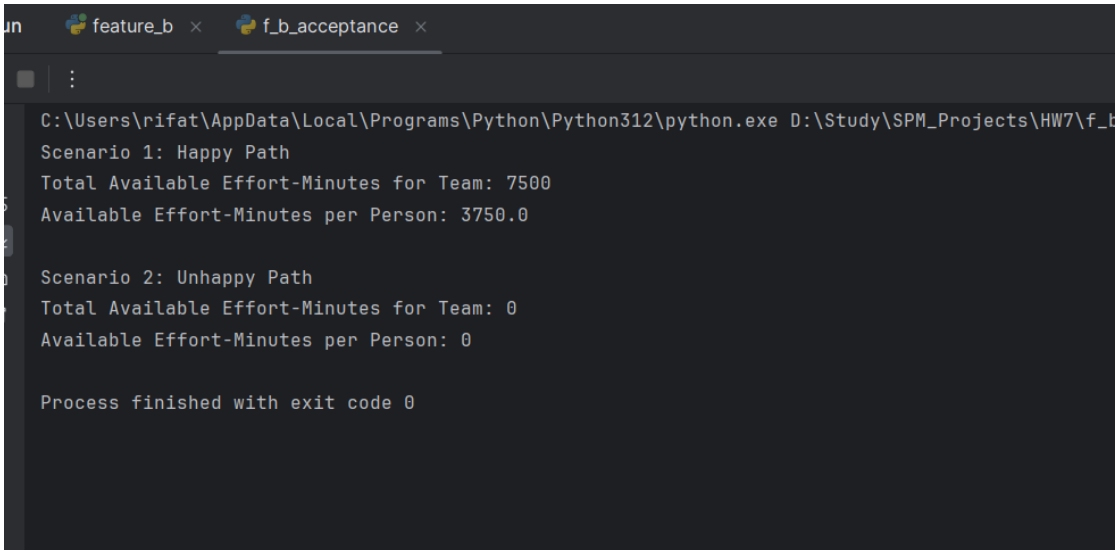
display_team_capacity(team_capacity_empty): Method to display the outcome, expected to show zero effort-hours.

Analysis of Scenarios:

Happy Path (Scenario 1): Creates a team with members and calculates the total and per person effort minutes. This scenario tests the system's ability to accurately calculate and display effort based on realistic input data, involving the creation, data manipulation, and result display methods.

Unhappy Path (Scenario 2): Tests the system with no team members added, validating its ability to correctly handle and display zero effort minutes in scenarios lacking input data, effectively utilizing the calculation and display methods without initial data manipulation.

Feature_b_acceptance_test:



Testing Format:

The testing format utilized here is a combination of scripted testing and assertion-based validation:

Scripted Testing: This format involves writing scripts that simulate specific scenarios or use cases of the application. It's particularly useful for features that include user

interaction, such as input and output operations, where the tester manually verifies the correctness of the application's behavior.

Assertion-Based Validation: Within the scripts, assertions are used where possible to automatically verify that the outcome of method calls matches the expected results. This adds a level of automated validation to the tests, ensuring that key functionalities behave as intended without manual verification.

Conclusion:

The meticulous approach to software testing detailed in this project exemplifies the fundamental role that structured and comprehensive testing plays in the software development lifecycle. It highlights the necessity of both unit and acceptance testing in uncovering and mitigating potential issues, thereby enhancing the software's quality and reliability. Ultimately, this testing strategy not only safeguards the functional integrity of the software but also ensures that it meets the end users' needs effectively, underscoring the indispensable value of thorough testing in achieving software excellence.