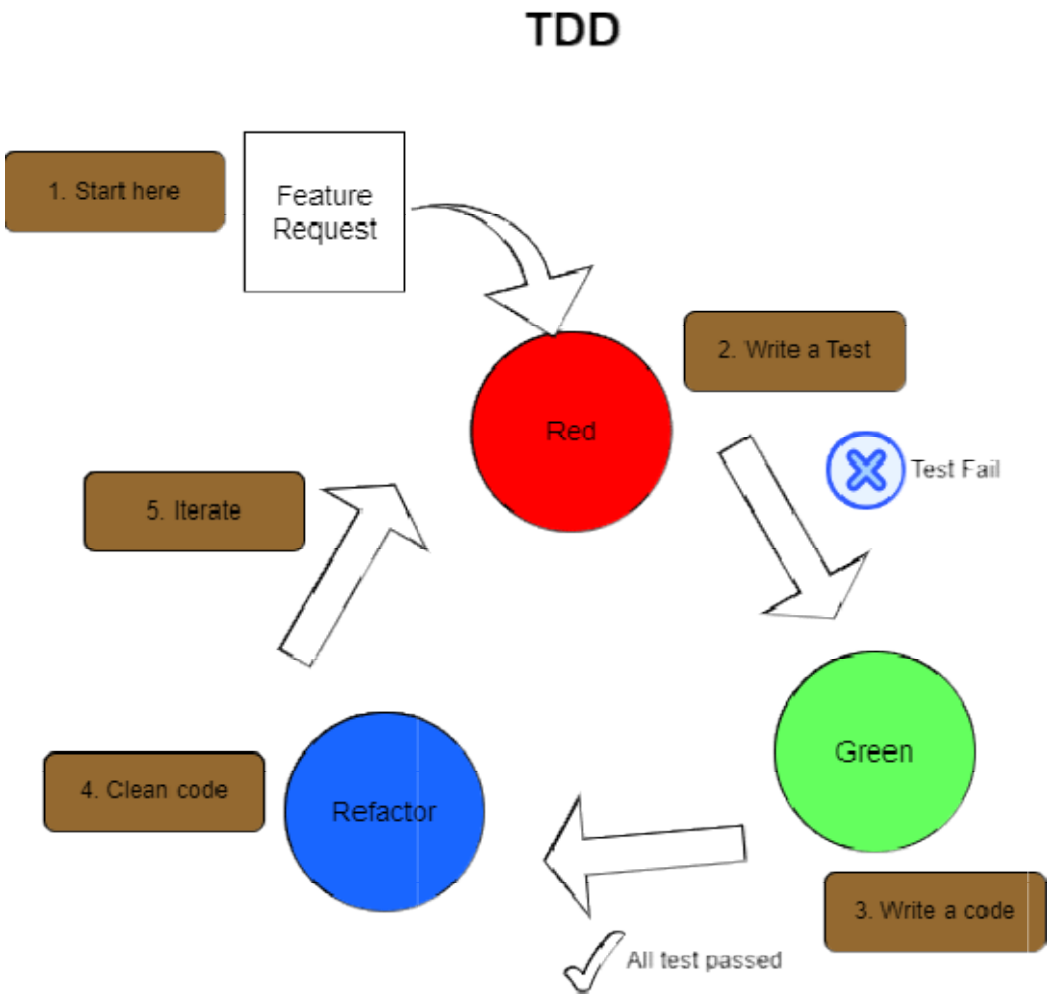
Assignment 1: Create an info graphic illustrating the Test-Driven Development (TDD) process. Highlight steps like writing tests before code, benefits such as bug reduction, and how it fosters software reliability.

TDD Info graphic diagram:



Steps in the Test-Driven Development (TDD) Process:

1. Read, understand, and process the feature or bug request.

- Start by thoroughly understanding the feature or bug report. This step sets the foundation for writing effective tests by ensuring a clear understanding of what needs to be implemented or fixed.

2. Translate the requirement by writing a unit test.

- Begin by writing a unit test that defines the expected behavior based on the requirement.

3. Writing tests before code encourages developers to think about the desired outcome upfront, promoting a clear understanding of the task at hand.

- This practice significantly reduces the likelihood of overlooking edge cases or requirements.

- Write and implement the code that fulfills the requirement.

- Write the minimum amount of code necessary to make the failing test pass.

- Focusing on passing the failing test ensures that the codebase remains lean and free from unnecessary features.

- By addressing the requirements in small, manageable increments, developers can maintain a high level of focus and productivity.

4. Clean up your code by refactoring.

- Once all tests pass, take the opportunity to refactor the code.

- Refactoring improves code quality, readability, and maintainability without changing its external behavior.

- By continuously refining the codebase, developers can reduce technical debt and ensure that the software remains adaptable to future changes.

5. Rinse, lather and repeat.

- Finally, repeat this cycle for each new feature or bug fix.

- Iterating through these steps ensures that the software is thoroughly tested, robust, and meets the specified requirements.

- Over time, this iterative process fosters software reliability by systematically addressing bugs and ensuring that new features are implemented with a high degree of accuracy.

Benefits such as bug reduction:

- By writing tests before implementing code, developers catch bugs earlier in the development process.

- Test-Driven Development (TDD) encourages thorough test coverage, which helps identify and fix issues before they become more complex and costly to resolve.
- With TDD, bugs are often caught at the unit level, making them easier to isolate and fix, reducing the risk of introducing regressions.

How it fosters software reliability:

- TDD promotes software reliability by ensuring that code is thoroughly tested throughout the development process.
- Since tests are written before code, developers have a clear target to aim for, resulting in more predictable and reliable outcomes.
- The iterative nature of TDD encourages developers to continuously validate and refine their code, leading to more robust and stable software.

In Real World Application,

Test-Driven Development (TDD) Process for Common Invoice Operations in Billing Applications:

1. Understanding the Requirement:

- The team receives a requirement to enhance the **billing application** to allow users to perform common invoice operations such as creating, updating, deleting and viewing invoices by their unique IDs.

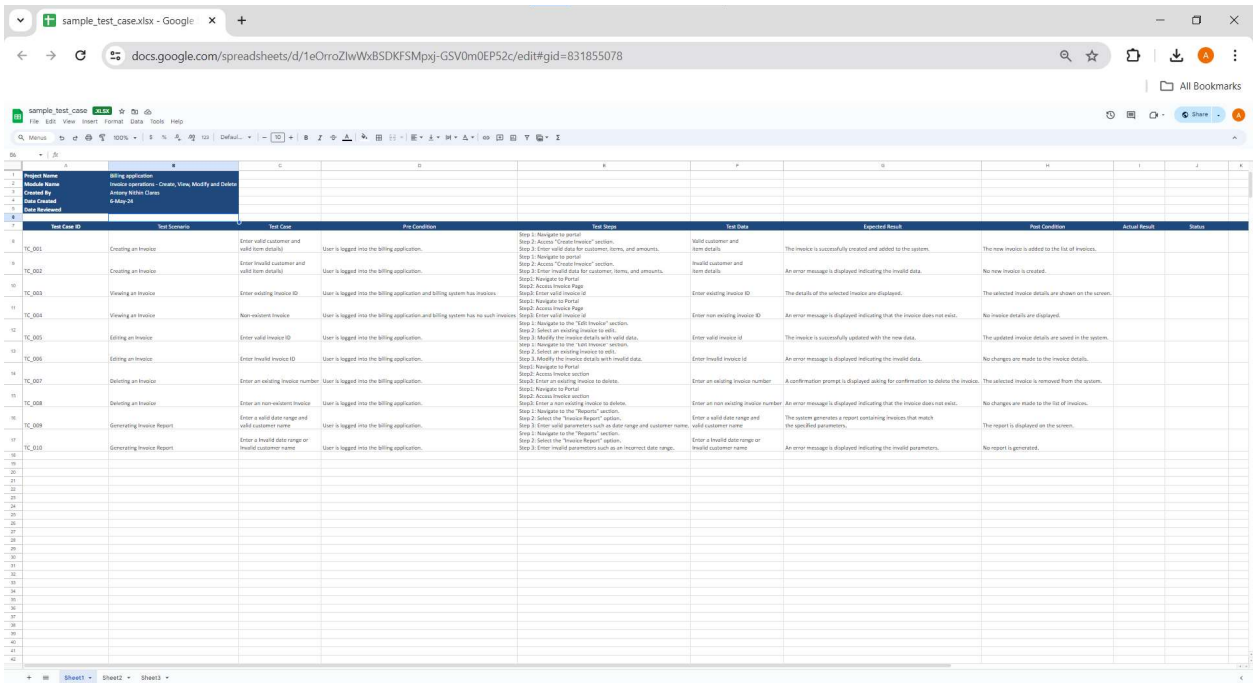
2. Writing Unit Tests:

- Define acceptance criteria for each common invoice operation, ensuring successful execution of tasks such as creating, updating, deleting, and viewing invoices.
- Develop unit test cases to validate the functionality of each common invoice operation, covering various scenarios including valid and invalid inputs, edge cases, and error conditions.

Test Case Scenario Excel File has been attached below,

https://docs.google.com/spreadsheets/d/1eOrroZIwWxBSDKFSMpxj-GSV0m0EP52c/edit?usp=sharing&ouid=102804329445938037247&rtpof=true&sd=true

Image of test case scenario:



Test Case ID	Test Scenario	Test Case	Pre Condition	Test Steps	Test Data	Expected Result
TC_001	Creating an Invoice	Enter valid customer and valid item details)	User is logged into the billing application.	Step 1: Navigate to portal Step 2: Access "Create Invoice" section. Step 3: Enter valid data for customer, items, and amounts.	Valid customer and item details	The invoice is successfully created and added to the system.
TC_002	Creating an Invoice	Enter Invalid customer and valid item details)	User is logged into the billing application.	Step 1: Navigate to portal Step 2: Access "Create Invoice" section. Step 3: Enter invalid data for customer, items, and amounts.	Invalid customer and item details	An error message is displayed indicating the invalid data.

3. Writing Code to Pass the Test:

- Modify the application's data storage and processing logic to support the execution of common invoice operations, including creating, updating, deleting, and retrieving invoices.
- Implement user interface changes to accommodate the execution of common invoice operations, providing appropriate user feedback and error handling mechanisms.
- Ensure the application correctly performs each common invoice operation according to the defined acceptance criteria, handling exceptions and errors gracefully.

4. Refactoring Code:

- Optimize data storage and processing mechanisms to enhance performance and scalability, ensuring efficient execution of common invoice operations.
- Enhance error handling and validation mechanisms to provide informative error messages and improve user experience during the execution of common invoice operations.

- Streamline code logic to improve readability, maintainability, and extensibility, making it easier to add new features or modify existing functionality in the future.

5. Repeating the Cycle:

- Iterate through writing tests, implementing code changes, and refactoring code as needed for each common invoice operation.
- Thoroughly test and validate the execution of all common invoice operations under various scenarios, ensuring robustness and reliability of the billing application.
- Follow TDD principles to drive the development process and maintain a high level of software quality and user satisfaction.

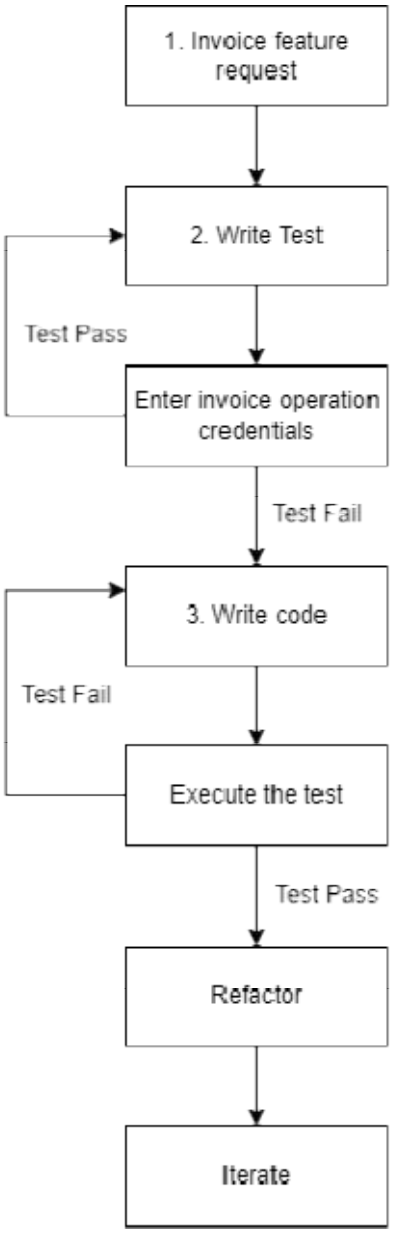
Enhanced Billing Invoice Operation - Bug Reduction Benefits:

- Preventive Testing: TDD's proactive approach catches bugs early, reducing their impact on invoice operations.
- Improved Code Quality: TDD's emphasis on small, testable units leads to cleaner, more bug-resistant code.
- Enhanced User Satisfaction: Fewer bugs mean a smoother experience for users interacting with the billing application.

Fostering Software Reliability in Billing Invoice Operations:

- Early Issue Identification: TDD detects potential problems in invoice operations early, allowing timely resolution.
- Thorough Functionality Validation: TDD rigorously tests each aspect of invoice operations, ensuring reliability.
- Continuous Integration: TDD promotes continuous testing, preventing regressions and maintaining reliability.

Flow Diagram of Invoice operations in Billing System:



By following this TDD process, the team can systematically develop and validate the common invoice operations within the billing application, ensuring that it meets user requirements, adheres to industry standards, and delivers a seamless user experience.