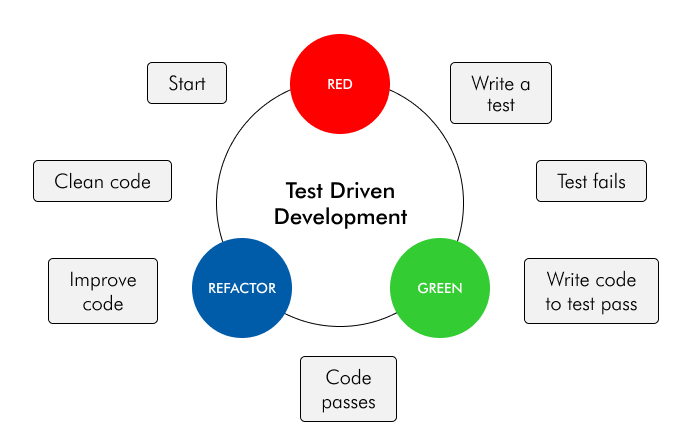
day3:

---------

Assignment 1: Create an infographic 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.

**Test Driven Development (TDD)** is a software development methodology that emphasizes writing **tests before writing the actual code**. It ensures that code is always tested and functional, reducing bugs and improving code quality. In TDD, developers write **small, focused tests** that define the desired functionality, then write the minimum code necessary to pass these tests, and finally, refactor the code to improve structure and performance.

This **cyclic process** helps in creating reliable, maintainable, and efficient software. By following TDD, teams can enhance **code reliability**, accelerate **development cycles**, and maintain high standards of **software quality**



Run all the test cases and make sure that the new test case fails.

1. **Red**: In this phase, you start by writing a test that defines the desired behavior or functionality of a specific piece of code. Initially, this test will fail because the corresponding code hasn’t been written yet. This failing test is often referred to as a “red” test.
2. **Green**: Once you have a failing test, your next step is to write the minimum amount of code necessary to make the test pass. This code may not be perfect or efficient; the goal is to satisfy the test’s conditions and make it pass. When the test passes, it becomes a “green” test, indicating that the desired functionality has been implemented.
3. **Refactor**: After making the test pass, you can improve the code’s design, structure, and efficiency while keeping the test green. Refactoring involves making changes to the code without changing its external behavior. The tests act as a safety net, helping you catch unintended side effects of your changes.

**Benefits of TDD:**Here are the benefits of Test-Driven Development (TDD):

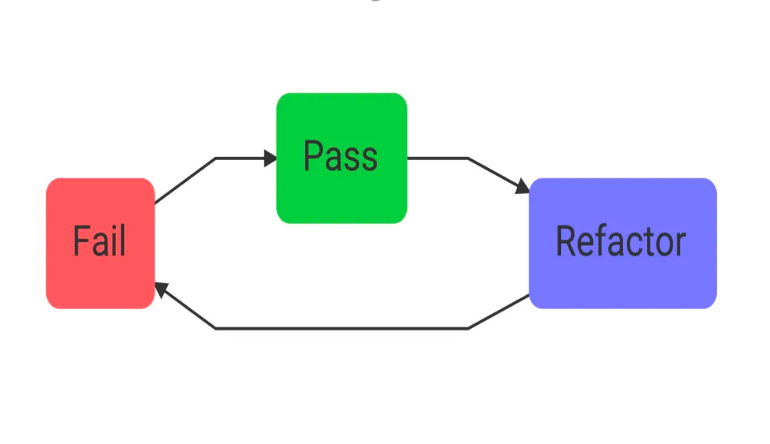
**1. Improved Code Quality:** TDD enforces a focus on writing clean, maintainable, and modular code from the outset. By writing tests first, developers must think critically about the design and architecture of their code, leading to higher code quality and fewer design flaws.

**2. Reduced Bugs and Defects:**With TDD, bugs and defects are identified early in the development process as tests are written before code implementation. This proactive approach helps catch issues before they propagate and become more challenging and costly to fix.

Assignment 2: Produce a comparative infographic of TDD, BDD, and FDD methodologies. Illustrate their unique approaches, benefits, and suitability for different software development contexts. Use visuals to enhance understanding.

**Test-Driven Development (TDD):**

Test-Driven Development (TDD) is a software development approach where tests are written before the actual code. The core idea is to write a test for a specific function or feature, which initially fails (the "red" phase), then write the minimal amount of code required to pass the test (the "green" phase), and finally refactor the code while ensuring it still passes the tests (the "refactor" phase). This cycle enhances the quality and reliability of code, ensuring each piece is thoroughly tested.



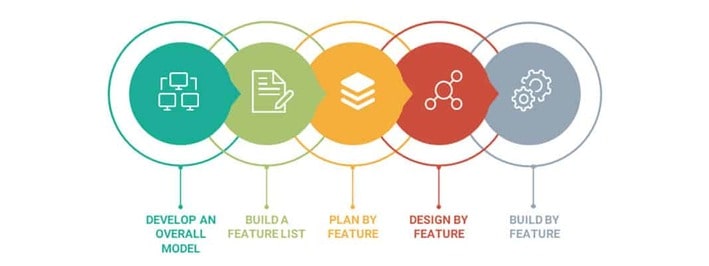
**Behavior Driven Development (BDD):**

BDD is an a[gile testing methodology](https://katalon.com/resources-center/blog/agile-testing-methodology) that uses system behavior to guide the development activities. Instead of starting with a test like in TDD, BDD starts with analyzing the desired behavior that developers want to create. After that, they’ll express the desired behavior using the **Gherkin syntax**, which consists of Given - When - Then statements. These statements show developers how to develop the code that fulfills the behaviors described.



**Feature-Driven Development(FDD):**

**FDD** stands for **Feature-Driven Development**. It is an agile iterative and incremental model that focuses on progressing the features of the developing software. The main motive of feature-driven development is to provide timely updated and working software to the client. In FDD, reporting and progress tracking is necessary at all levels.



**Differences between TDD, FDD and BDD**

|  |  |  |  |
| --- | --- | --- | --- |
| **Methodologies** | **TDD** | **BDD** | **FDD** |
| Introduction | A software development process where tests are written before code. | An extension of TDD focusing on system behavior from the end user's perspective. | An iterative and incremental software development methodology focusing on feature delivery. |
| Unique Approaches |  Write a test for a small piece of functionality.   Implement the code to pass the test.   Refactor the code while keeping the test passing.   **Cycle:** Red (fail) -> Green (pass) -> Refactor. |  Define behavior in plain language (Gherkin).   Write scenarios and acceptance criteria.   Implement code to meet the specified behavior.   **Cycle:** Specification -> Test -> Code -> Validate. |  Develop an overall model.   Build a feature list.   Plan by feature.   Design by feature.   Build by feature.   **Cycle:** Domain Modeling -> Feature List -> Planning -> Design -> Implementation. |
| Benefits |  Ensures code reliability.   Facilitates refactoring.   Improves code design.   Encourages simple, clean code. |  Aligns development with business needs.   Enhances communication among stakeholders.   Improves user satisfaction.   Provides clear documentation. |  Focuses on delivering tangible, working features.   Scales well for larger teams.   Encourages iterative development.   Provides clear milestones. |
| Suitability |  Suitable for complex, logic-heavy applications.   Ideal for projects requiring high code quality.   Useful in continuous integration environments. |  Suitable for projects with strong stakeholder involvement.   Ideal for user-centric applications.   Beneficial for projects needing clear requirements and communication. | * Suitable for large-scale projects. * Ideal for teams with a mix of junior and senior developers. * Useful for projects requiring frequent deliveries and feature tracking. |
| Visuals and Diagrams | A flowchart illustrating Red -> Green -> Refactor. | A flowchart illustrating Specification -> Test -> Code -> Validate. | A flowchart illustrating Domain Modeling -> Feature List -> Planning -> Design -> Implementation. |