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 practice that focuses on creating unit test cases before developing the actual code. It is an iterative approach combining programming, unit test creation, and refactoring.

- The TDD approach originates from the Agile manifesto principles and Extreme programming.
- In TDD, developers create small test cases for every feature based on their initial understanding. The primary intention of this technique is to modify or write new code only if the tests fail. This prevents duplication of test scripts.

Three Phases of Test-Driven Development:

1.Create precise tests: Developers need to create exact unit tests to verify the functionality of specific features. They must ensure that the test compiles so that it can execute. In most cases, the test is bound to fail. This is a meaningful failure as developers create compact tests based on their assumptions of how the feature will behave.

2.Correcting the Code: Once a test fails, developers must make the minimal changes required to update the code to run successfully when re-executed.

3.Refactor the Code: Once the test runs successfully, check for redundancy or any possible code optimizations to enhance overall performance. Ensure that refactoring does not affect the external behaviour of the program.

Benefits of TDD:

- **Bug Reduction:**

  Catch and fix bugs early in the development process.

- **Software Reliability:**

  Ensure the software behaves as expected and remains stable over time.

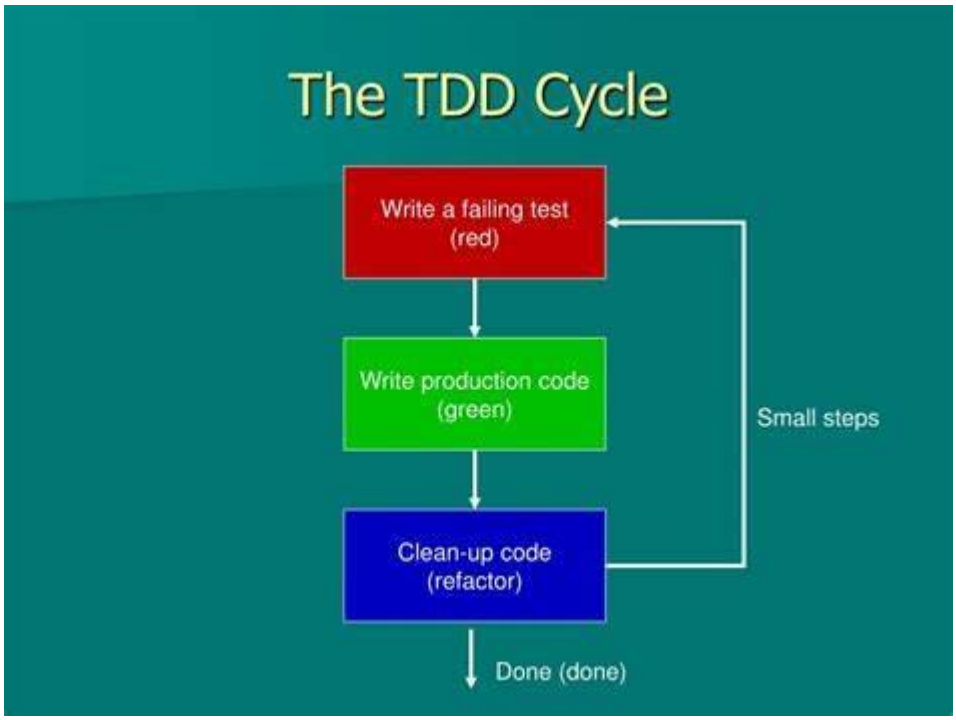
- **Improved Code Design:**

Encourage better design and modular, maintainable code.

- **Faster Debugging:**

  Quickly identify and resolve issues with the help of automated tests.

TDD CYCLE



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.

1. Overview Section

Title: TDD vs. BDD vs. FDD: Understanding Key Software Development Methodologies

2. Unique Approaches

TDD (Test-Driven Development)

Approach: Write tests before writing the corresponding code.

Cycle: Red-Green-Refactor.

Focus: Ensuring each small piece of code works correctly.

BDD (Behavior-Driven Development)

Approach: Define behavior in plain language (using Given-When-Then format).

Cycle: Define behavior -> Write tests -> Write code.

Focus: Ensuring the software behaves as expected from the user's perspective.

FDD (Feature-Driven Development)

Approach: Develop features incrementally based on user-defined functionality.

Cycle: Domain walkthrough -> Build feature list -> Plan by feature -> Design by feature -> Build by feature.

Focus: Delivering visible, working features regularly.

3. Benefits

TDD

Code Quality: High due to thorough testing.

Refactoring: Easier with confidence due to existing tests.

Documentation: Tests serve as documentation for code functionality**.**

BDD

Collaboration: Improves collaboration between technical and non-technical stakeholders.

Requirements: Clearer understanding of requirements.

User Focus: Ensures development aligns with user needs.

FDD

Scalability: Scales well for large projects.

Visibility: Regular delivery of features provides progress visibility.

Client Satisfaction: Continuous client involvement and feedback.

4. Suitability for Different Contexts

TDD

Best For: Projects requiring high code quality and maintainability.

Examples: Libraries, APIs, and mission-critical systems.

BDD

Best For: Projects needing clear communication between stakeholders.

Examples: User-centric applications, startups, and Agile environments.

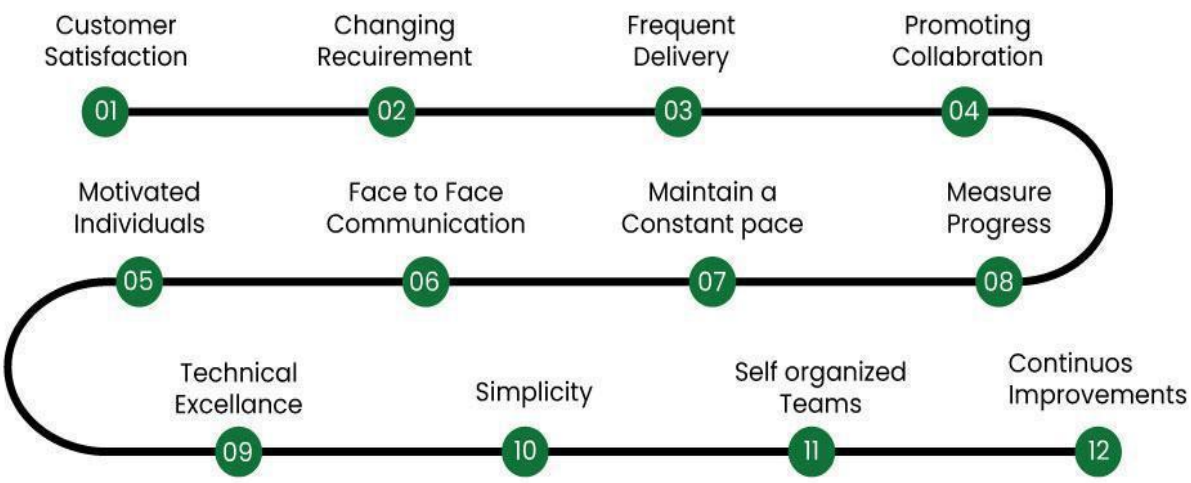
FDD

Best For: Large-scale projects needing clear structure and regular deliveries.

Examples: Enterprise applications, long-term projects with multiple teams.

3.Agile Principles:

There are 12 agile principles mentioned in the Agile Manifesto. Agile principles are guidelines for flexible and efficient software development. They emphasize frequent delivery, embracing change, collaboration, and continuous improvement. The focus is on delivering value, maintaining a sustainable work pace, and ensuring technical excellence.



12 Principles of Agile Methodology

The Agile Alliance defines twelve lightness principles for those who need to attain agility:

1. Our highest priority is to satisfy the client through early and continuous delivery of valuable computer software.

2. Welcome dynamic necessities, even late in development. Agile processes harness modification for the customer's competitive advantage.

3. Deliver operating computer software often, from a pair of weeks to a couple of months, with a preference to the shorter timescale.

4. Business individuals and developers should work along daily throughout the project.

5. The build comes around actuated people. offer them the setting and support they have, and trust them to urge the task done.

6. the foremost economical and effective methodology of conveyancing info to and among a development team is face-to-face speech.

7. Working with computer software is the primary life of progress.

8. Agile processes promote property development. The sponsors, developers, and users will be able to maintain a relentless pace indefinitely.

9. Continuous attention to technical excellence and smart style enhances nimbleness.

10. Simplicity—the art of maximizing the number of work not done—is essential.

11. the most effective architectures, necessities, and styles emerge from self–organizing groups.

12. At regular intervals, the team reflects on a way to become simpler, then tunes and adjusts its behaviour consequently.