Day-4-Final_Assignment

Date: 30/07/2024

By: Parth Gurval

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.

Ans ->

The Test-Driven Development (TDD) Process:

What is TDD?

- Test-Driven Development (TDD) is a software development approach where tests are written before the code that needs to pass them.
- This process ensures that the code meets the requirements from the start.

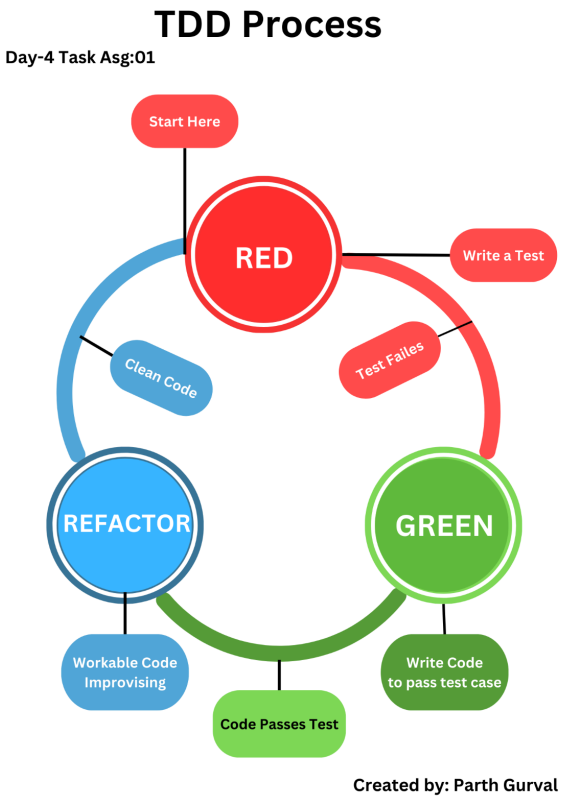
The TDD Cycle

1. **Red Phase (Red Circle):**
    - **Start Here**: Begin the cycle by writing a test.
    - **Write a Test**: Define a test for the new feature.
    - **Test Fails**: Run the test to see it fail, confirming the feature is not yet implemented.

2. **Green Phase (Green Circle):**
    - **Write Code to Pass Test Case**: Implement the necessary code to pass the test.
    - **Code Passes Test**: Run the test again to ensure it passes with the new code.
3. **Refactor Phase (Blue Circle):**

- **Workable Code Improvising**: Refactor the code to improve its design and structure.
- **Clean Code**: Ensure the code is clean and maintainable.

TDD Infographic View:



Start Here:

- **Description**: This is where the TDD process begins. You start by identifying the functionality you want to implement.
- **Action**: Initiate the TDD cycle.

Write a Test:

- **Description**: Write a test for the new functionality you plan to add.
- **Action**: Define the expected behavior and outcome for the new feature in the form of a test.

Test Fails:

- **Description**: Run the test you just wrote. Since the functionality is not yet implemented, the test should fail.
- **Action**: Execute the test to confirm that it fails, which ensures the test is valid and detects the missing feature.

Write Code to Pass Test Case:

- **Description**: Write the minimal amount of code needed to make the test pass.
- **Action**: Implement just enough code to fulfill the requirements specified by the test.

Code Passes Test:

- **Description**: Run the test again to verify that the new code passes the test.
- **Action**: Ensure that the test is successful, indicating that the new functionality works as intended.

Workable Code Improvising (Refactor):

- **Description**: Refactor the code to improve its structure and readability without changing its behavior.
- **Action**: Clean up and optimize the code to make it more maintainable.

Clean Code (Refactor):

- **Description**: After refactoring, ensure the code is clean, well-organized, and adheres to best practices.
- **Action**: Finalize the refactoring process, resulting in high-quality code.

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.

Ans:

TDD (Test-Driven Development), BDD (Behavior-Driven Development), and FDD (Feature-Driven Development) are methodologies that can be used within the framework of Agile and the Software Development Life Cycle (SDLC).

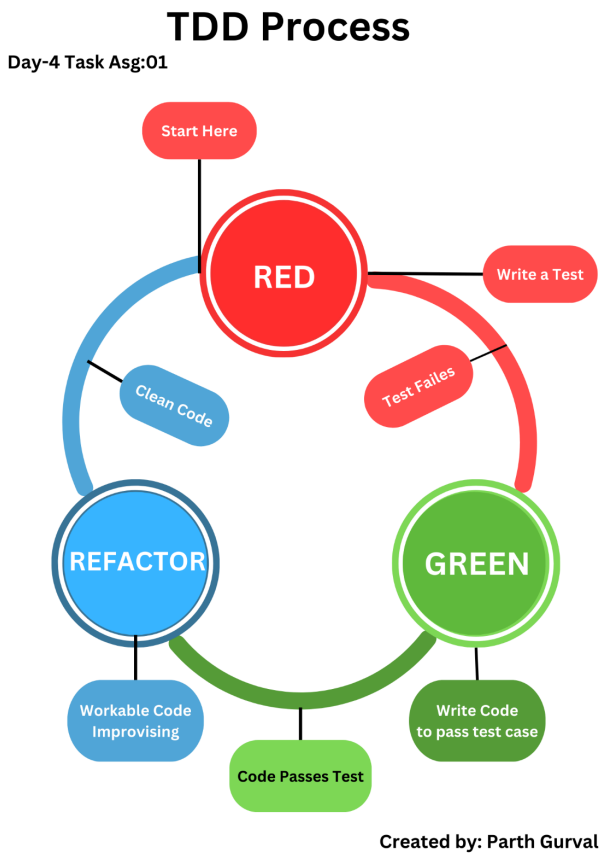
1. **TDD (Test-Driven Development):**
   - **Concept**:
     - TDD is a development process where developers write tests for a feature before writing the code to implement that feature.
     - The cycle involves writing a failing test, writing just enough code to pass the test, and then refactoring the code.
     - TDD can be incorporated into various phases of the SDLC, particularly during the development and testing phases, to ensure high-quality code and early detection of defects.

- **Unique Approaches**
  - Write a test:
    - Before writing any functional code, a test is written that defines the expected behavior of a small piece of functionality.
  - Run the test (it should fail):
    - The test is executed, and it is expected to fail. This step confirms that the test is correctly detecting the absence of the required functionality.

- Write code to pass the test:
  - This step focuses on implementing the functionality required to satisfy the test conditions.

- Refactor the code:
  - Refactoring ensures that the codebase remains clean and efficient.

- **TDD Process Infographic Image:**



- **The TDD Cycle:**

1. **Red Phase (Red Circle)**
   - **Start Here**: Begin the cycle by writing a test.
   - **Write a Test**: Define a test for the new feature.

- **Test Fails**: Run the test to see it fail, confirming the feature is not yet implemented.

2. **Green Phase (Green Circle)**
   - **Write Code to Pass Test Case**: Implement the necessary code to pass the test.
   - **Code Passes Test**: Run the test again to ensure it passes with the new code.

3. **Refactor Phase (Blue Circle)**
   - **Workable Code Improvising**: Refactor the code to improve its design and structure.
   - **Clean Code**: Ensure the code is clean and maintainable

- ## Benefits:

  - **Early Bug Detection**:
    - By writing tests before the code, bugs are identified and fixed at an early stage, reducing the cost and effort of fixing them later.
    - This proactive approach to testing ensures that defects are caught as soon as they are introduced.

  - **High Code Quality**:
    - TDD promotes writing small, testable units of code, which leads to higher code quality.
    - Each unit of code is thoroughly tested, ensuring that it performs as expected.

  - **Better Design and Modularity**:
    - The need to write tests first encourages developers to think about the design and architecture of the code before implementation.
    - This leads to more modular and decoupled code, which is easier to maintain and extend.

- ## Suitability:

- **Best for Projects Where Code Reliability and Quality Are Paramount**:

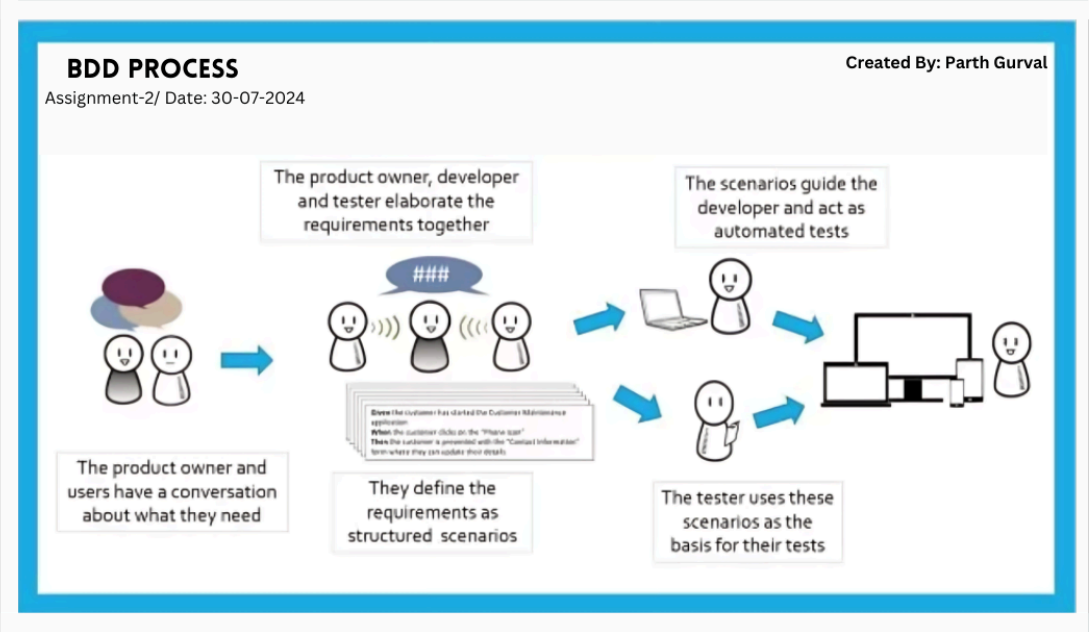
    - TDD is particularly suitable for projects that require high reliability and quality, such as financial systems, healthcare software, and mission-critical applications.

    - It ensures that each piece of code is validated through tests, reducing the likelihood of defects and ensuring that the software behaves as expected.

2. BDD (Behavior-Driven Development):

- **Concept**:
    - BDD extends TDD by focusing on the behavior of an application from the end-user's perspective.
    - It uses a shared language to write scenarios that describe the behavior in a way that both technical and non-technical stakeholders can understand.
    - BDD integrates into the SDLC by influencing requirements gathering, development, and testing phases.
    - It helps ensure that requirements are well-understood and that tests are aligned with business goals.
- **Unique Approaches of BDD:**
    - **Approach:**
    - **Define Behaviors Using Gherkin Syntax (Given-When-Then)**:
        - Behaviors are defined using a structured language called Gherkin, which uses the format Given-When-Then.
    - **Write Scenarios as Tests**:
        - Scenarios are written based on the defined behaviors, serving as executable specifications.
    - **Implement Code to Fulfill the Behavior**:
        - Code is written to implement the behavior described in the scenarios.

- **BDD Process Infographic Image:**



The cycle involves:

a. It defining requirements as structured scenarios.
b. We use these scenarios to guide development and testing.
c. This ensures the developed software meets user needs and expectations.

BDD can be incorporated into various phases of the SDLC, particularly during the requirements gathering, development, and testing phases, to ensure alignment with business goals.

- ## The BDD Cycle
    - **The Product Owner and Users Have a Conversation About What They Need**:
        - This step involves initial discussions to understand the requirements from the user's perspective.

    - **The Product Owner, Developer, and Tester Elaborate the Requirements Together**:
        - Collaborative sessions to detail the requirements and ensure a shared understanding among all stakeholders.

    - **They Define the Requirements as Structured Scenarios**:
        - Requirements are converted into scenarios, typically using a structured language like Gherkin.

    - **The Scenarios Guide the Developer and Act as Automated Tests**:
        - These scenarios serve as a guide for developers and are used to write automated tests.

    - **The Tester Uses These Scenarios as the Basis for Their Tests**:
        - Testers use the defined scenarios to create and run tests, ensuring that the implementation meets the specified behavior.

- ## Benefits:

    - **Improved Collaboration**:
        - Shared understanding of requirements leads to better communication and fewer misunderstandings.

    - **Clear Specifications**:
        - This helps in creating a single source of truth for the desired behavior of the system.

    - **Alignment with User Requirements**:
        - Scenarios are written based on user stories, ensuring that the implemented features meet user needs.

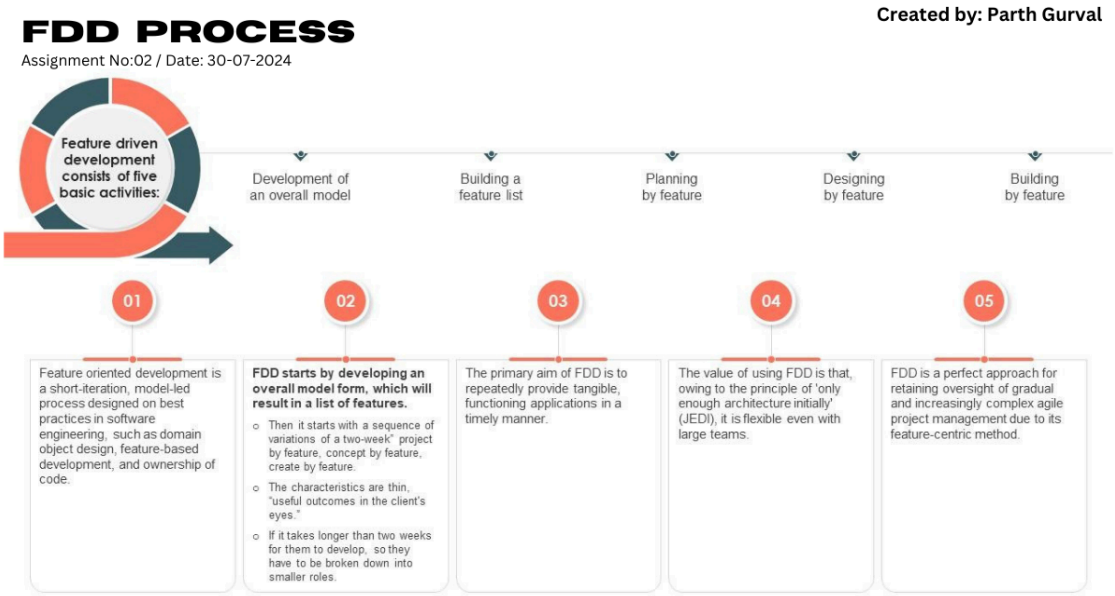
- **Suitability:**
  - **Ideal for Projects That Require Close Collaboration and Clear Communication Among Diverse Teams**:
    - BDD is well-suited for projects where communication between technical and non-technical team members is crucial.
    - It is particularly beneficial for projects with complex requirements that need to be clearly understood by all stakeholders.

3. FDD (Feature-Driven Development)

- **Concept:**
    - FDD is an Agile methodology focused on delivering tangible, working software features in short iterations.
    - It involves developing an overall model, building a feature list, planning by feature, designing by feature, and building by feature.
    - FDD aligns with the SDLC by structuring the development process around features.
    - It involves detailed planning, designing, and building of features, making it suitable for projects where feature delivery is a primary focus.

- **Unique Approaches of FDD:**
    - **Approach:**
    - **Develop an Overall Model**:
        - An overall model of the system is developed to understand the scope and structure of the project.
    - **Build a Features List**:
        - A comprehensive list of features is created based on the overall model.
    - **Plan by Feature**:
        - Features are prioritized and planned for development in iterations.
    - **Design by Feature**:
        - The design process ensures that all aspects of the feature are well-understood and planned.
    - **Build by Feature**:
        - Features are developed and integrated into the system one by one.

- **FDD Process Infographic Image:**



- **The cycle involves**:
  - Developing an overall model.
  - Building a feature list.
  - Planning by feature.
  - Designing by feature.
  - Building by feature.

- ## The FDD Cycle
- **Developing an Overall Model**:
    - This step involves creating a high-level model of the system to understand its structure and components.
    - Teams collaborate to develop a shared understanding of the system architecture.

- **Building a Feature List**:
    - Identify and prioritize a list of features that need to be developed.
    - Each feature represents a small, client-valued function that can be developed and delivered incrementally.

- **Planning by Feature**:
    - Create a detailed plan for developing each feature.
    - This includes estimating effort, assigning tasks, and scheduling work.

- **Designing by Feature**:
    - Design the feature in detail, considering how it will be integrated into the overall system.
    - Ensure that the design aligns with the overall model and architectural guidelines.

- **Building by Feature**:
    - Implement the feature according to the design.
    - Test the feature to ensure it meets the specified requirements and integrates well with the existing system.

- ## Benefits:
    - **Timely Delivery of Features**:
        - FDD focuses on delivering features in a timely manner, ensuring that client needs are met quickly.

    - **Client-Centric Approach**:
        - Features are defined based on client value, ensuring that the development process is aligned with client priorities.

    - **Manageable Progress Tracking**:
        - FDD provides a clear framework for tracking progress based on feature completion.

- ## Suitability:
- **Suitable for Large, Complex Projects Where Timely Feature Delivery and Client Feedback Are Crucial**:
    - FDD is ideal for large projects with complex requirements that need to be delivered incrementally.
    - It is particularly beneficial for projects where client feedback is essential for guiding development.

Assignment 3: Agile Project Planning - Create a one-page project plan for a new software feature using Agile planning techniques.

Include backlog items with estimated story points and a prioritized list of user stories.

Ans:

- **Project Name: Construction Material Management Software**

- **Project Description:**
  - This software solution to simplify the management of orders, inventory, and financial transactions for construction material businesses, enhancing the customer experience through a convenient platform for order placement, payment, and tracking.

- **Prioritized Backlog Items with Estimated Story Points:**

1. **User Roles and Permissions** (Epic)
   - Owner: Manage purchase and selling of materials (Invoices) (8 story points)
   - Owner: Handle Worker accounts, worker salaries, and customer accounts (5 story points)
   - Owner: Monitor daily sales and purchases (3 story points)
   - Manager: View and manage (CURD) orders (8 story points)
   - Worker: View assigned orders and tasks (3 story points)
   - Customer: Place orders online or in-store (5 story points)

2. **Inventory Management** (Epic)
   - Comprehensive listing of construction materials (10 story points)
   - Real-time inventory tracking and updates (8 story points)
   - Updating Inventory according to needs (5 story points)

3. **Order Management** (Epic)
   - Facilitate order placement and processing for both in-store and online purchases (10 story points)
   - Track order status and provide real-time GPS updates (8 story points)

4. **Financial Management** (Epic)

    - Monitor and manage daily sales and purchases and sales (8 story points)
    - Track due bills and customer debts (5 story points)
5. **Customer Experience** (Epic)
    - Online portal for easy order placement and bill management (10 story points)
    - Integration of payment gateways for bill payments (8 story points)

6. **Notifications and Reminders** (Epic)
    - Automated reminders for due bills, pending tasks, Inventory, and order status updates (8 story points)

7. **Invoice** (Epic)
    - Includes all data-related billing like customer details, order details, shipment details, GST/non-GST bills, payment options, etc. (10 story points)

8. **Login and Sign Up** (Epic)
    - Sign Up: Allow new users to create an account with their details (5 story points)
    - Login: Enable registered users to log in securely to access their respective dashboards and functionalities (5 story points)

- **Prioritized List of User Stories:**

    a. As an Owner, I want to manage purchase and selling of materials (Invoices) so that I can track my business transactions.

    b. As a Manager, I want to view and manage (CURD) orders so that I can ensure timely order fulfillment.

    c. As a Customer, I want to place orders online or in-store so that I can easily purchase construction materials.

    d. As an Owner, I want to monitor daily sales and purchases so that I can track my business performance.

    e. As a Worker, I want to view assigned orders and tasks so that I can plan my work schedule.

Sprint 1:

- User Roles and Permissions (Owner, Manager, Worker, Customer)
- Inventory Management (comprehensive listing of construction materials)
- Order Management (facilitate order placement and processing)

Sprint 2:

- Financial Management (monitor and manage daily sales and purchases and sales)
- Customer Experience (online portal for easy order placement and bill management)
- Notifications and Reminders (automated reminders for due bills, pending tasks, Inventory, and order status updates)

Sprint 3:

- Invoice (includes all data-related billing like customer details, order details, shipment details, GST/non-GST bills, payment options, etc.)
- Login and Sign Up (Sign Up: Allow new users to create an account with their details, Login: Enable registered users to log in securely to access their respective dashboards and functionalities)

Assignment 4: Daily Standup Simulation - Write a script for a Daily Standup meeting for a development team working on the software feature from Assignment

1. Address a common challenge and incorporate a solution into the communication flow.

Ans:

Daily Standup Meeting

Team Members:

- John (Scrum Master)
- Sarah (Developer)
- Michael (Developer)
- Emily (Developer)
- David (QA Engineer)

John (Scrum Master): Good morning, team. Let's get started with our daily standup meeting. Remember, we're working on the Construction Material Management Software. Our goal is to deliver a high-quality product that meets our customer's needs.

Sarah (Developer): Hi, John. Yesterday, I worked on the User Roles and Permissions feature. I completed the development of the Owner role, which includes managing purchase and selling of materials. Today, I plan to work on the Manager role.

Michael (Developer): Hi, John. I worked on the Inventory Management feature. I completed the comprehensive listing of construction materials. However, I faced a challenge with the real-time inventory tracking and updates. I'm struggling to integrate the API with our system.

Emily (Developer): Hi, John. I worked on the Order Management feature. I completed the facilitation of order placement and processing. Today, I plan to work on the track order status and provide real-time GPS updates.

David (QA Engineer): Hi, John. I worked on testing the User Roles and Permissions feature. I found a few bugs, which I've reported to Sarah. Today, I plan to test the Inventory Management feature.

John (Scrum Master): Great updates, team. Michael, can you elaborate on the challenge you're facing with the real-time inventory tracking and updates?

Michael (Developer): Yes, John. The API we're using is not providing the expected output. I've tried to troubleshoot the issue, but I'm not sure how to resolve it.

Sarah (Developer): I think I can help Michael. I've worked with a similar API before. Let me take a look at the code.

John (Scrum Master): Great teamwork, Sarah. Michael, please share the code with Sarah, and let's schedule a meeting for later today to discuss the solution.

Michael (Developer): Sounds good, John. Thank you, Sarah.

John (Scrum Master): Alright, team. Let's review our progress and plan for the day. We're on track to meet our sprint goals. Keep up the good work, and let's address any challenges that come our way.

Common Challenge:

- Integrating APIs with the system

Solution:

- Team collaboration and knowledge sharing
- Scheduling a meeting to discuss the solution

Communication Flow:

- Open discussion of challenges and solutions
- Collaboration among team members
- Review of progress and plan for the day