Agile can be implement in many ways ---- The primary focus of xp framework is – programming (code can be used again & again) , scrum is deliver projects, kanban is manage process.

## Describe the stages of the “Red, Green, Refactor” cycle with a suitable example

### Meaning of Red–Green–Refactor Cycle

The **Red–Green–Refactor** cycle is the core workflow of **Test-Driven Development (TDD)**.
In this approach, tests are written **before** the actual code.

#### Red Phase

- Write a **test for a new feature**
- Run the test
- The test **fails** because the feature is not implemented yet

Red indicates that the required functionality is missing.

#### Green Phase

- Write the **minimum code** required to pass the test
- Focus only on correctness, not design
- All tests should pass

Green indicates that the feature works as expected.

#### Refactor Phase

- Improve code structure
- Remove duplication
- Enhance readability and design
- Ensure tests still pass

Refactoring improves **quality without changing behavior**.

---

### Example: Calculator Addition

**Requirement:** Add two numbers

**Red phase**

- Write a test expecting `add(2,3) = 5`
- Test fails because method is not implemented

**Green phase**

- Implement simple `add()` method
- Test passes

**Refactor phase**

- Improve method naming or structure
- Remove unnecessary code

simple java code for calculation of discount for refactoring

## before refactoring

```
public double calculateDiscount(double amount) {
    double discount = 0;

    if (amount > 1000) {
        discount = amount * 0.1;
    } else if (amount > 500) {
        discount = amount * 0.05;
    }

    return discount;
}
```

## After refactoring

```
public double calculateDiscount(double amount) {
    return amount * getDiscountRate(amount);
}

private double getDiscountRate(double amount) {
    if (amount > 1000) {
        return 0.10;
    }
    if (amount > 500) {
        return 0.05;
    }
    return 0.0;
}
```

## Explain the impact of the Green phase in TDD

### Role of Green Phase in TDD

The **Green phase** ensures that the written test passes by implementing the **simplest possible solution**.

### Impact of Green Phase

1. **Ensures Functional Correctness**
    - Confirms that the requirement is met
    - Code behaves as expected
2. **Prevents Over-Engineering**
    - Developers write only what is needed
    - Avoids unnecessary features
3. **Builds Confidence**
    - Passing tests give immediate feedback
    - Developers know the code works
4. **Enables Safe Refactoring**
    - Green tests act as a safety net
    - Changes can be made without fear
5. **Supports Incremental Development**
    - Small working steps
    - Faster progress tracking

---

### Example

- Login validation test fails (Red)
- Simple validation logic implemented (Green)
- Now system accepts valid input correctly

# Explain the Agile approach and demonstrate Agile techniques with a real-world scenario

## What Agile Is NOT

- Agile is **not** long-term requirements gathering
- Agile avoids months of documentation upfront

## What Agile IS

> **Agile is an iterative, incremental approach that focuses on early delivery, continuous feedback, and adapting to change.**

Agile values:

- Working software
- Customer collaboration
- Frequent feedback
- Responding to change

---

## Real-World Scenario: Online Food Delivery App

### Traditional Approach

- Gather all requirements for months
- Build entire system
- Release after long delay
- High risk of failure

### Agile Approach

#### Agile Techniques Used:

1. **User Stories**
    - “As a user, I want to order food”
    - “As a user, I want to track delivery”
2. **Iterations**
    - Iteration 1: Login & restaurant list
    - Iteration 2: Order placement
    - Iteration 3: Payment & tracking
3. **Frequent Releases**
    - Basic app released early

    - Users start using it
4. **Customer Feedback**
    - Users request faster checkout
    - Team adapts in next iteration
5. **Continuous Testing**
    - Automated tests ensure quality
    - Bugs fixed early

-----------------------------------------------------------------------------------------------------------

## What Is a User Story?

A **user story** is a short, simple description of a feature written from the **user's point of view**.

It describes **what the user wants** and **why**, not how to build it.

**Definition**

> A user story is a lightweight requirement that captures a user need in simple language to support iterative development and customer collaboration.

---

## 2. Standard Format of a User Story

Most user stories follow this template:

> **As a** `<type of user>`
> **I want** `<some goal>`
> **So that** `<some benefit>`

### Example

> As a **student**,
> I want to **submit assignments online**,
> so that **I don't miss deadlines**.

---

## 3. Why Agile Uses User Stories (Instead of Big Requirements)

Traditional approach:

- Long requirement documents
- Months of analysis
- Hard to change

Agile approach:

- Short user stories
- Continuous discussion
- Easy to change

**User stories encourage conversation, not paperwork.**

---

---

## 4. User Stories vs Requirements

| Aspect | User Stories | Traditional Requirements |
|---|---|---|
| Length | Short | Long documents |
| Focus | User value | System details |
| Flexibility | High | Low |
| Documentation | Lightweight | Heavy |
| Change handling | Easy | Difficult |

---

## 5. Who Writes User Stories?

- **Product Owner / Customer** usually writes them
- Developers and testers **refine** them together
- Stories are discussed continuously

Stories are **collaborative**, not imposed.

---

## 6. Acceptance Criteria

Each user story has **acceptance criteria** that define when it is complete.

### Example

User story:

> As a user, I want to log in securely.

Acceptance criteria:

- Valid username & password → login success
- Invalid credentials → error message
- Password masked on screen

Acceptance criteria turn stories into **testable work**.

---

## 7. User Stories in Real-World Agile Project (Example)

### Scenario: Online Shopping App

Some user stories:

- As a user, I want to search products
- As a user, I want to add items to cart
- As a user, I want to make payment

Development:

- Iteration 1: Search + product view
- Iteration 2: Cart + checkout
- Iteration 3: Payment + tracking

Each iteration delivers **working features**.

---

## 8.User Stories and Testing

- Stories drive **Test-Driven Development (TDD)**
- Acceptance tests are written from stories
- “Done” means story + tests are complete

XP belief:

> If you can’t test it, it’s not a good story.

## 9. Advantages of User Stories

- Simple and easy to understand
- Focus on user value
- Encourage communication
- Support frequent change
- Fit well with iterative development

---

# Version Control Terminology

---

## Repository

### Meaning

> The **repository** is the **central storage** where all project files and their **complete history** are stored.

- Lives on a version control server (GitHub, GitLab, SVN)
- Each project has **one repository**
- Acts as the **single source of truth**

### Example

- A GitHub repo for a **college mini-project**
- Contains:
    - Code
    - Tests
    - Commit history

### Real-life analogy

**Library –** stores all books and editions.

---

## Sandbox (Working Copy)

### Meaning

> A **sandbox** is a **local copy** of the repository on a developer's machine where they work safely.

- Also called *working copy*
- Each developer has **their own sandbox**
- Never shared with others

### Example

- You clone a Git repo to your laptop
- You write and test code locally

### Real-life analogy

**Photocopy of a book** you can write on without affecting the original.

---

## Check Out

### Meaning

> **Check out** means creating a sandbox by copying files from the repository.

- First step before development
- Some systems also lock files (older systems)

### Example

```
git clone https://github.com/project/repo.git
```

### Real-life analogy

Borrowing a book from the library to read at home.

---

## Update

### Meaning

> **Update** brings the **latest changes** from the repository into your sandbox.

- Keeps your local copy up-to-date
- You can also update to an **older version**

### Example

```
git pull
```

### Real-life analogy

Updating an app to the latest version.

---

## Lock

### Meaning

> A **lock** prevents others from editing a file while you are working on it.

- Mostly used in **older version control systems**

- Rare in Git (Git prefers merging)

### Example

- Locking a design document so only one person edits it

### Real-life analogy

Locking a shared notice board while writing.

---

## Check In / Commit

### Meaning

> **Check in (commit)** saves your changes from the sandbox into the repository.

- Creates a permanent history record
- Should be small and meaningful

### Example

```
git commit -m "Add login validation"
```

### Real-life analogy

Saving your assignment to Google Drive.

---

## Revert

### Meaning

> **Revert** discards your local changes and restores files to the last saved state.

- Used when local code is broken
- Faster than debugging sometimes

### Example

```
git checkout .
```

### Real-life analogy

Clicking **Undo** to discard recent edits.

---

## Tip / Head

### Meaning

**Tip (or Head)** refers to the **latest committed version** in the repository.

- Updating gives you the tip
- Changes with branches

### Example

- Latest commit on `main` branch

### Real-life analogy

Latest edition of a textbook.

---

## Tag / Label

### Meaning

A **tag** marks a **specific point in history** for easy reference.

- Used for releases
- Does not change over time

### Example

```
git tag v1.0
```

### Real-life analogy

Bookmarking a chapter for future reference.

---

## Roll Back

### Meaning

**Roll back** removes a bad commit and returns the repository to an earlier state.

- Used when a change causes serious problems
- Differs by version control system

### Example

- Removing a faulty commit that broke production

### Real-life analogy

Rolling back software to a previous stable version.

---

## Branch

### Meaning

A **branch** is a separate line of development with its own history.

- Used for:
    - Experiments
    - Features
    - Bug fixes

### Example

```
git branch feature-login
```

### Real-life analogy

A side road branching off a main highway.

---

## Merge

### Meaning

**Merge** combines changes from different branches or developers into one.

- Conflicts may occur
- Conflicts must be resolved manually

### Example

```
git merge feature-login
```

### Real-life analogy

Combining two edited versions of the same document.

---

## Summary Table

| Term | Meaning |
|---|---|
| Repository | Central storage |

| **Term** | **Meaning** |
|---|---|
| Sandbox | Local working copy |
| Check out | Create sandbox |
| Update | Get latest changes |
| Commit | Save changes |
| Revert | Discard local changes |
| Tip/Head | Latest version |
| Tag | Mark a version |
| Roll back | Remove bad change |
| Branch | Parallel development |
| Merge | Combine changes |