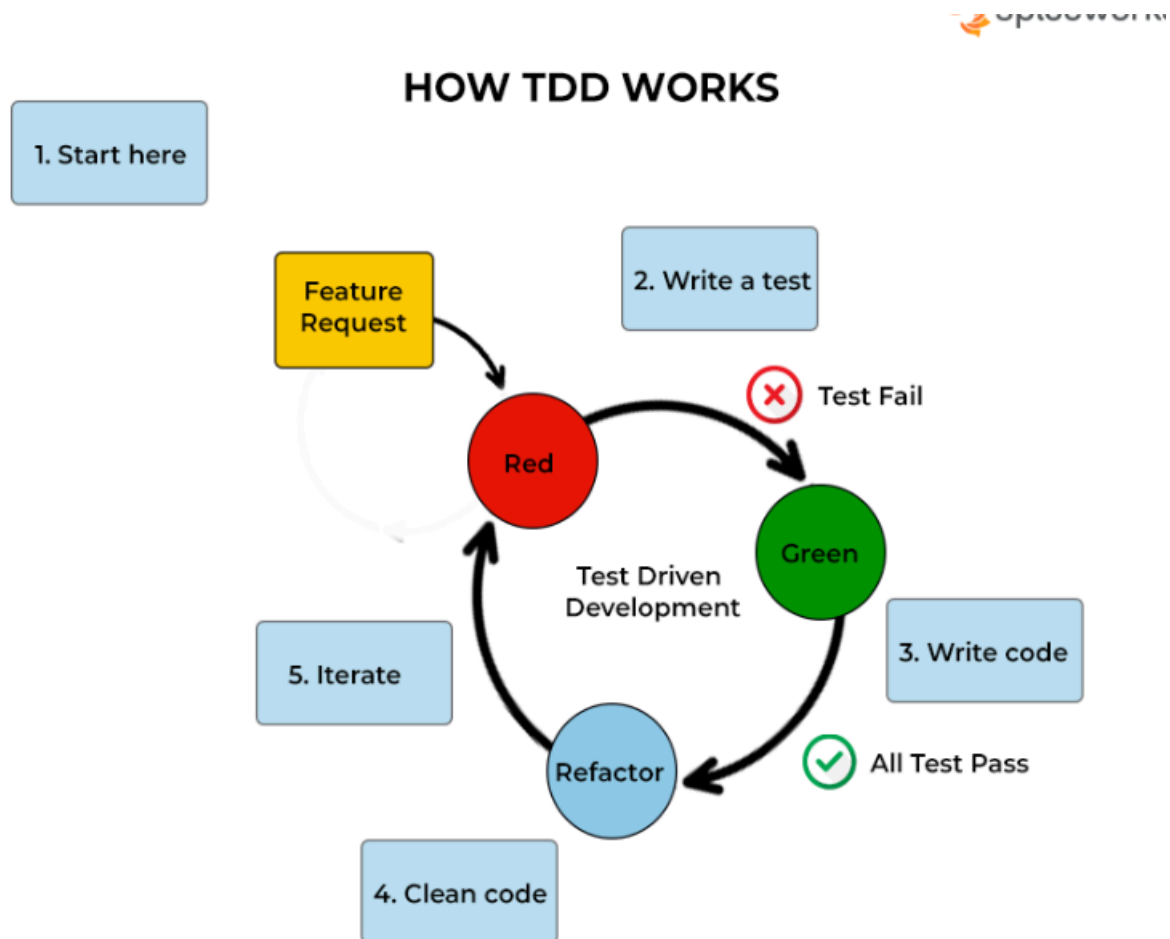


Assignment 1: Create an infographic illustrating the Test-Driven Development (TDD) process. Highlight steps like writing tests before code, benefits such as bug reduction, and how it fosters software reliability.

Ans:



1. Write Tests

- Developers write small, specific automated tests based on user stories or requirements.

2. Run Tests

- Execute the tests to ensure they fail initially, indicating that there is no code to fulfill the requirements yet.

3. Write Code

- Develop the minimum code necessary to pass the failing tests. This code is often referred to as the "implementation code."

4. Run Tests Again

- Execute the tests again. They should pass now as the minimal implementation code has been written.

5. Refactor Code

- Optimise and refactor both the implementation code and the test code to improve readability, maintainability, and performance.

Benefits of Test-Driven Development (TDD):

- Bug Reduction: TDD helps catch and fix bugs early in the development process, reducing the likelihood of bugs reaching production and minimising overall debugging time.
- Improved Software Reliability: By continuously testing and verifying the functionality of the codebase, TDD fosters software reliability and ensures that the software behaves as expected.

- **Faster Development:** Despite the initial investment in writing tests, TDD can lead to faster development cycles by encouraging developers to focus on delivering only the necessary functionality and preventing time-consuming debugging later on.

Conclusion:

Test-Driven Development (TDD) promotes a disciplined and iterative approach to software development, where tests drive the implementation and code quality is paramount. By following the TDD process, developers can create more reliable, maintainable, and bug-free software products.

Illustration of a developer writing tests before writing code:

Writing Test :

```
1  package com.wipro;
2
3  import static org.junit.Assert.assertEquals;
4
5
6
7
8  class SumTest {
9
10     @Test
11
12     public void testAdd() {
13         Sum sum = new Sum();
14         assertEquals(5, sum.add(2,3));
15     }
16
17 }
```

Writing Code:

Sum.java X SumTest.java

```
1 package com.wipro;
```

```
org/src/com/wipro/Sum.java
```

```
3 public int add(int a,int b) {
```

```
4     return a+b;
```

```
5 }
```

```
6 }
```

```
7
```

```
1 package com.wipro;
2
3 import static org.junit.Assert.assertEquals;
4
5
6
7
8 class SumTest {
9
10     @Test
11
12     public void testAdd() {
13         Sum sum = new Sum();
14         assertEquals(5, sum.add(2,3));
15     }
16
17 }
18
```

Problems Servers Terminal Data Source Explorer

Finished after 0.261 seconds

Runs: 1/1 Errors: 0

> SumTest [Runner: JUnit 5] (0.010 s)

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:

1. Test-Driven Development (TDD)

- Approach:

- Develop tests prior to implementation code.
- Test coverage aligns with specific requirements or user stories.

- Benefits:

- Early identification and prevention of bugs.
- Enhanced code quality and design.
- Guarantees code is both testable and maintainable.

Suitability:

- Suitable for projects with precise and well-defined requirements.
- Effective for small, iterative development phases.

Developer writing tests before code:

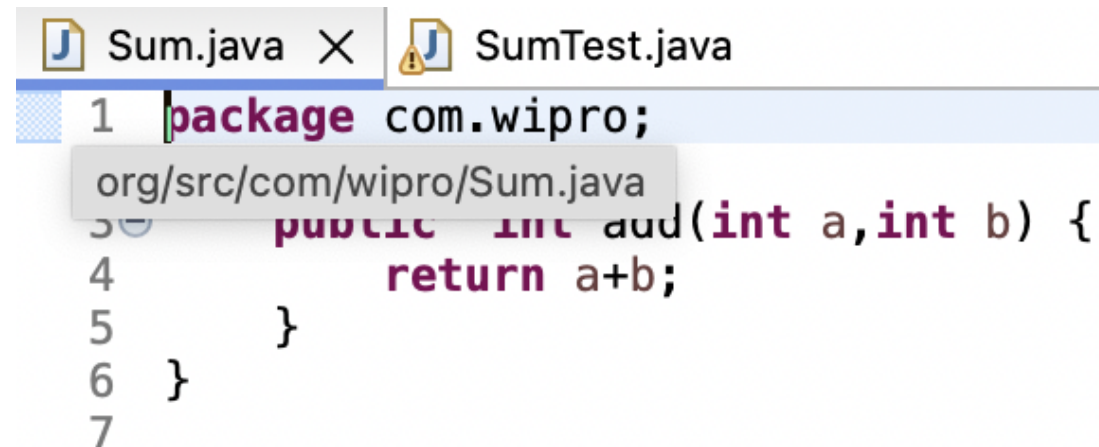
Writing Test:

```

1  package com.wipro;
2
3  import static org.junit.Assert.assertEquals;
4
5
6
7
8  class SumTest {
9
10     @Test
11
12     public void testAdd() {
13         Sum sum = new Sum();
14         assertEquals(5, sum.add(2,3));
15     }
16
17 }

```

Writing Code:



```

Sum.java
1  package com.wipro;
2
3  public int add(int a,int b) {
4      return a+b;
5  }
6  }
7

```


Sum.java SumTest.java X

```
1 package com.wipro;
2
3 import static org.junit.Assert.assertEquals;
4
5
6
7
8 class SumTest {
9
10 @Test
11
12     public void testAdd() {
13         Sum sum = new Sum();
14         assertEquals(5, sum.add(2,3));
15     }
16
17 }
18
```

Problems Servers Terminal Data Source Explorer

Finished after 0.261 seconds

Runs: 1/1 Errors: 0

>  SumTest [Runner: JUnit 5] (0.010 s)

2. Behavioural Driven Development:

- Approach:

- Centres around defining user behaviour through user stories.
- Utilises natural language specifications to articulate desired behaviour.
- Tests are framed from the viewpoint of end-users.

Benefits:

1. Promotes teamwork among developers, testers, and stakeholders.
2. Enhances communication and clarifies requirements.
3. Fosters a customer-focused development approach.

Suitability:

- Ideal for projects with intricate business logic and diverse stakeholder needs.
- Beneficial for teams implementing continuous integration and delivery practices.

Team discussing user stories:

Feature: User Authentication

In order to access personalised content,

As a user,

I want to be able to log in to my account.

Scenario: Successful login with valid credentials

Given the user is on the login page

When the user enters a valid username and password

Then the user should be redirected to the dashboard

And the user should see a welcome message

Scenario: Failed login with invalid credentials

Given the user is on the login page

When the user enters an invalid username and/or password

Then the user should see an error message

And the user should remain on the login page

3. FDD (Feature Driven Development)

- Approach:

- Emphasises iterative feature delivery.
- Breaks features into smaller, more manageable tasks.
- Prioritises domain modelling and feature importance.

Benefits:

1. Emphasises organising development around features.
2. Advocates for delivering work in small, iterative cycles to gather feedback.
3. Offers a structured way to manage project scope and track progress.

- Suitability:

- Ideal for handling large-scale projects with intricate needs.
- Beneficial for teams prioritising feature delivery and client contentment.

Team working on feature breakdown:

```

package com.wipro;

public class AudioPlayer {
    private boolean isPlaying;
    private int currentPlaybackTime;

    public void load(String audioFile) {
        System.out.println("Loading audio file: " + audioFile);
    }

    public void play() {
        System.out.println("Playing audio");
        isPlaying = true;
    }

    public void pause() {
        System.out.println("Pausing audio");
        isPlaying = false;
    }

    public void stop() {
        System.out.println("Stopping audio");
        isPlaying = false;
        currentPlaybackTime = 0;
    }

    public void displayCurrentTime() {
        System.out.println("Current playback time: " + currentPlaybackTime);
    }
}

```

Conclusion:

Each software development methodology provides distinct strategies for creating software, tailored to varying project requirements and team structures. Selecting the appropriate methodology hinges on considerations like project intricacy, team magnitude, and stakeholder needs. Irrespective of the methodology chosen, the emphasis should be on fostering collaboration, effective communication, and delivering meaningful outcomes to users.