Name: Swarnali Ghosh

Java Pre-Skilling Training Session

Assignment -2.2

Module-2

Mail-id: swarnalighosh666@gmail.com

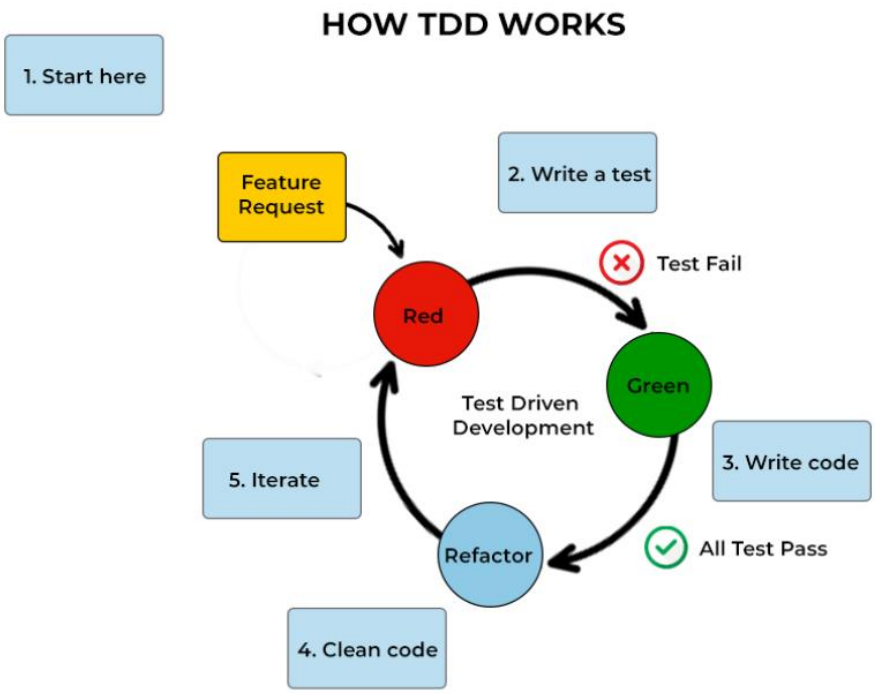
ASSIGNMENT-1

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.

SOLUTION:

TDD (Test Driven Development)

Test-driven development (TDD) is defined as an iterative methodology that prioritizes the creation of and checking against test cases at every stage of software development, by converting each component of the application into a test case before it is built and then testing and tracking the component repeatedly.



TDD Cycle

1. **Write a Test**
   - Write a test for a new feature using JUnit.

```
@Code Test
public void testAddition() {
    Calculator calculator = new Calculator();
    int result = calculator.add(2, 3);
    assertEquals(5, result);
}
```

2. **Run the Test**
   - Run the test to see it fail, ensuring the feature is not yet implemented.

Example Output:

testAddition FAILED (Expected: 5, Actual: 0)

3. Write Code

- Write the minimal code necessary to pass the test.

Example:

```
public class Calculator {
   public int add(int a, int b) {
      return a + b;
   }   }
```

4. Run Tests Again

- Run all tests using JUnit to ensure the new code passes.

Example Output:

testAddition PASSED

5. Refactor Code

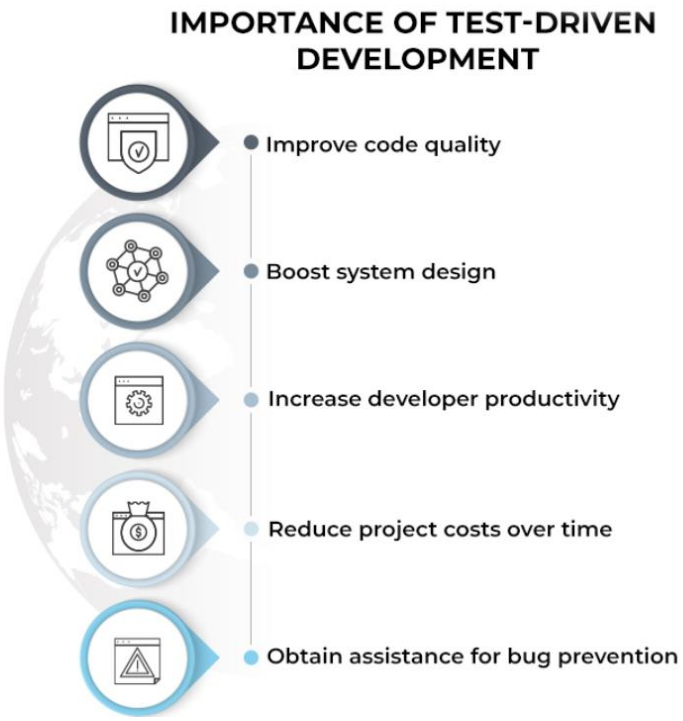
- Refactor the new code to improve its structure and readability.

Example (no changes needed for simple addition):

```
public class Calculator {
   public int add(int a, int b) {
      return a + b;
   } }
```

6. Repeat

- Continue the cycle for each new feature or functionality.



Benefits of TDD

- **Bug Reduction**
  - Identify and fix bugs early, reducing defects.
  - **Example**: Detecting off-by-one errors early.
- **Code Quality**
  - Encourages clean, maintainable code through continuous refactoring.
  - **Example**: Regularly updating method names and structures for clarity.
- **Fosters Software Reliability**
  - Comprehensive test coverage ensures stable software.
  - **Example**: Confidence in making changes without breaking existing functionality.
- **Documentation**
  - Tests act as documentation for code behavior.
  - **Example**: Tests provide usage examples and expected results.
- **Improved Design**

    - Leads to better design decisions by thinking through requirements first.
    - **Example**: Designing methods that are easier to test and use.

Best Practices for TDD in Java

- **Keep Tests Small and Focused**
    - Each test should focus on a small piece of functionality.
    - **Example**: Testing a single method's behavior.
- **Write Clear and Descriptive Test Names**
    - Test names should clearly describe their purpose.
    - **Example**: testAddition_PositiveNumbers
- **Test Edge Cases**
    - Ensure tests cover edge cases and potential errors.
    - **Example**: Testing with zero, negative numbers, and large values.
- **Integrate TDD with CI/CD**
    - Use CI/CD to run tests automatically on commit.
    - **Example**: Using Jenkins or GitHub Actions for automated testing.
- **Use Mocking Libraries**
    - Use libraries like Mockito to mock dependencies.
    - **Example**: Mocking a database connection in tests.

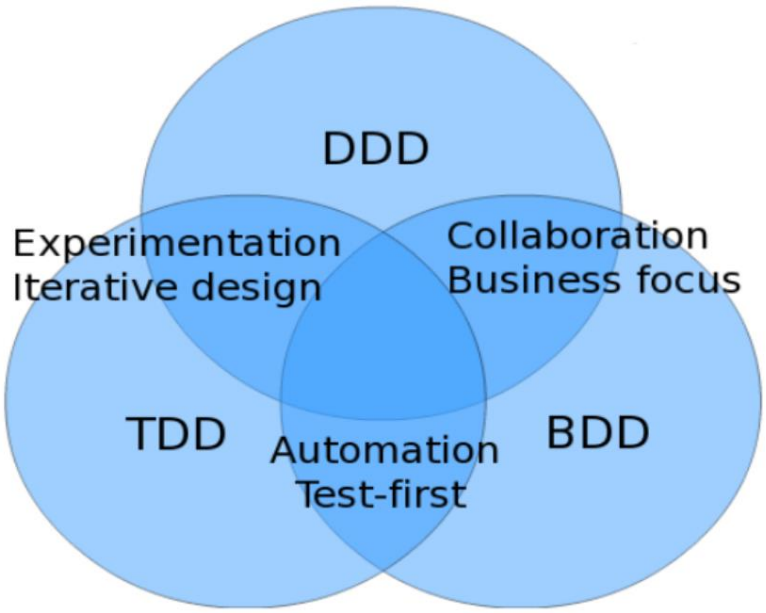
Tools for TDD in Java

- **Testing Frameworks**
    - JUnit
    - TestNG
- **Mocking Libraries**
    - Mockito
    - EasyMock
- **Build Tools**
    - Maven
    - Gradle

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.

SOLUTION:



Test-Driven Development (TDD)

Definition:

TDD is a development methodology where tests are written before the code. The cycle involves writing a test, making it pass with minimal code, and then refactoring.

Benefits:

- Ensures code is thoroughly tested from the start.

- Facilitates better design and architecture.
- Provides immediate feedback on code quality.
- Reduces bugs and enhances reliability.

Suitability:

- Best for projects where code quality and test coverage are critical.
- Suitable for developers who are comfortable writing tests first.
- Ideal for maintaining legacy systems and complex applications.

Real-Life Example:

A team is developing a new feature for a banking app that calculates interest on savings accounts. Before writing the actual interest calculation code, the developers first write tests to define the expected output for various input values. Only after these tests are written and failing do they proceed to implement the interest calculation logic, ensuring that the feature works as intended and meets the predefined criteria.

Behavior-Driven Development (BDD)

Definition:

BDD extends TDD by focusing on the behavior of the software from an end-user perspective. It uses natural language constructs to write test scenarios, promoting collaboration between developers, testers, and non-technical stakeholders.

Benefits:

- Enhances collaboration between non-technical stakeholders and developers.
- Clarifies requirements through clear and understandable specifications.
- Provides comprehensive documentation of system behavior.

Suitability:

- Ideal for projects requiring strong collaboration between business and technical teams.
- Useful for developing user-centric applications.
- Effective in Agile environments with frequent iterations and changes.

Real-Life Example:

A product team is building an e-commerce website. They start by writing scenarios in plain language, like "Given a user is on the checkout page, when they enter valid payment details, then the payment is processed successfully." These scenarios are used to guide the development and ensure that the end product meets user expectations, facilitating communication between business analysts, developers, and QA testers.

Feature-Driven Development (FDD)

Definition:

FDD is an iterative and incremental software development methodology focused on delivering features. It involves creating a model, building a feature list, planning by feature, designing by feature, and building by feature.

Benefits:

- Focuses on delivering tangible, working software frequently.
- Scalable for large teams and projects.
- Provides clear visibility and progress tracking.

Suitability:

- Suitable for large-scale projects with many features.
- Best for organizations with a need for detailed planning and tracking.
- Effective in environments where requirements are well-understood.

Real-Life Example:

A software development company is creating a customer relationship management (CRM) system. They start by modeling the overall system and then create a feature list that includes functionalities like "Add new customer," "Update customer details," and "Generate customer reports." Each feature is designed and implemented in separate iterations, ensuring that the system grows in a controlled and predictable manner, with frequent deliveries of working software.

Comparisons between TDD,BDD and FDD are as follows:-

Criteria	TDD	BDD	FDD
Focus	Test-first development	Behavior specification	Feature delivery
Process	Red -> Green -> Refactor	Given -> When -> Then	Model -> Feature List -> Plan -> Design -> Build
Collaboration Level	Mainly developer-driven	High collaboration (business and technical)	Medium (cross-functional teams)
Documentation	Code and tests	User stories and scenarios	Feature list and design documents
Best Use Cases	High code quality, legacy systems	User-centric, Agile projects	Large-scale projects, clear requirements