Assignment :1

Task 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.

TDD:

- A software development process where tests are written before the code.
- Ensures that the software works as expected from the beginning.

The TDD Cycle

1. Write a Test
   - Write a simple test for a new function or feature.
   - The test should be clear and focused on a single aspect of the functionality.

2. Run the Test
   - Run the test to see it fail.
   - This confirms that the test is working correctly and the feature isn't already implemented.

3. Write the Code
   - Write the minimum amount of code required to pass the test.
   - Keep the implementation as simple as possible.

4. Run All Tests
   - Run all the tests to ensure the new code doesn't break existing functionality.
   - This step confirms that the new code integrates well with the existing codebase.

5. Refactor the Code
   - Refactor the code to improve its structure and readability without changing its behavior.
   - Ensure that all tests still pass after refactoring.

6. Repeat
   - Continue this cycle for each new feature or improvement.

Benefits of TDD

Bug Reduction

- Early detection and fixing of bugs.
- Each new piece of functionality is thoroughly tested before integration.

-Improved Code Quality

- Encourages writing simpler, more modular, and cleaner code.
- Continuous refactoring leads to better software design.

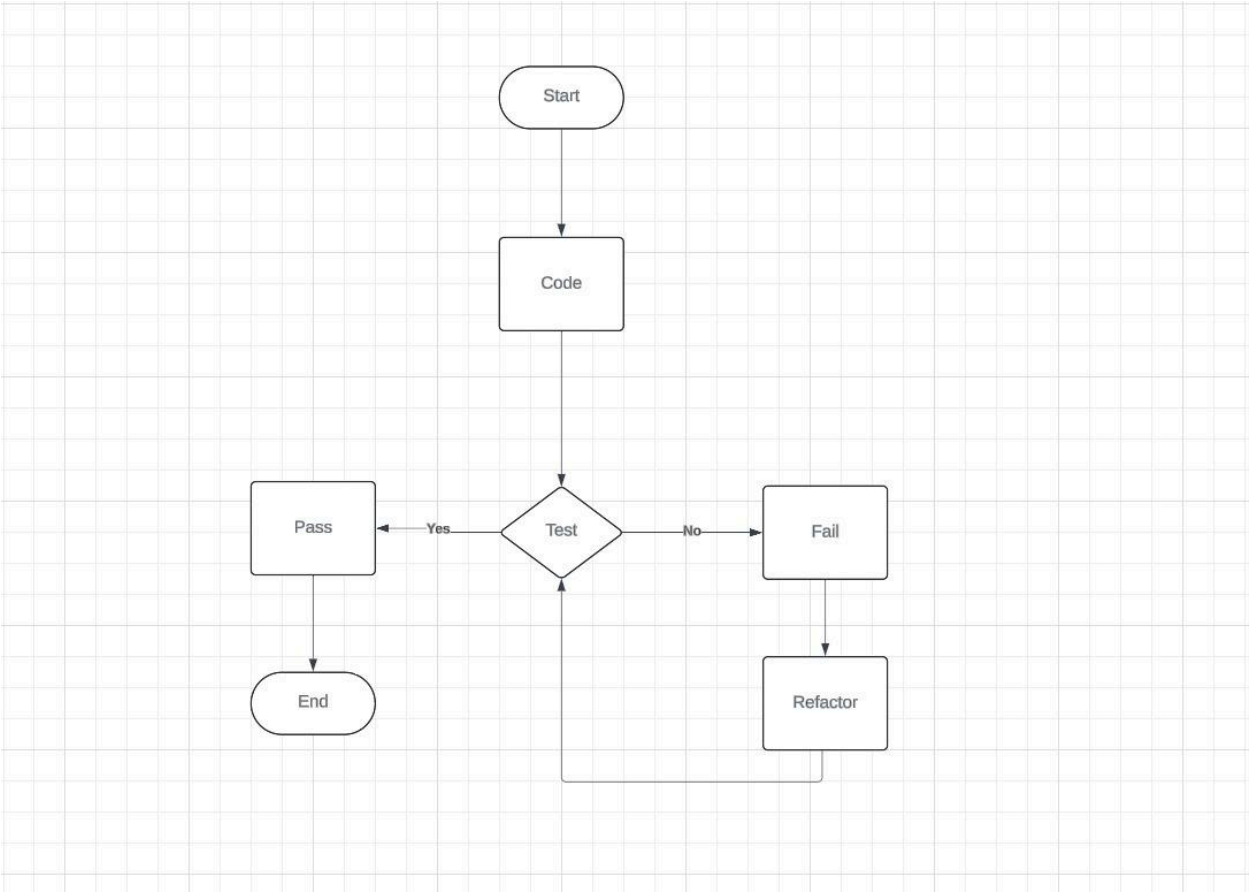
Faster Development
- Reduces the time spent on debugging.
- Clear test cases provide a clear understanding of requirements and functionality.

Fosters Reliability
- Provides a safety net that ensures changes don't introduce new bugs.
- Confidence in the software's behavior under various scenarios.

Documentation
- Tests serve as living documentation for the code.
- Easy to understand what the code is supposed to do.

Visual Representation

Flowchart:
- Test > Fail > Code > Pass > Refactor > Repeat



Graphs/Charts:

- Bug Reduction: A line graph showing a decrease in bugs over time.
- Development Speed: A bar chart showing faster development phases.
- Code Quality: A quality rating scale improving over iterations.

Assignment :2

Task 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.Test-Driven Development (TDD)
-Approach:
- Write tests before writing the actual code.
- Focuses on creating test cases for every small functionality.

Steps:
- Write a test.
- Run the test and see it fail.
- Write the minimum code to pass the test.
- Run all tests to confirm.
- Refactor the code.
- Repeat.

Benefits:
- Early bug detection.
- High code quality and maintainability.
- Ensures all functionalities are tested.

Suitability:
- Projects requiring high reliability.
- Complex systems with a need for continuous integration.

2. Behavior-Driven Development (BDD)
Approach:
- Write tests based on the expected behavior of the software.
- Uses natural language constructs (Given-When-Then).
- Encourages collaboration between developers, testers, and non-technical stakeholders.

Steps:
- Define a feature using a story format.
- Write scenarios based on behavior.
- Implement steps to pass the scenarios.
- Run the scenarios to validate behavior.

Benefits:
- Ensures software meets business requirements.
- Improves communication among stakeholders.
- Facilitates user-centric development.

Suitability:
- Projects with clear business requirements.
- Agile environments with frequent iterations.
- Teams with diverse technical and non-technical members.

3.Feature-Driven Development (FDD)**

Approach:
- Develops software by focusing on features.
- Iterative and incremental process.
- Emphasizes client-valued functionality.

Steps:
- Develop an overall model.
- Build a feature list.
- Plan by feature.
- Design by feature.
- Build by feature.

Benefits:
- Clear focus on delivering client-valued features.
- Scalable for large teams.
- Efficient for projects with well-defined features.

Suitability:
- Large-scale projects.
- Teams requiring clear feature prioritization.
- Environments needing predictable delivery timelines.

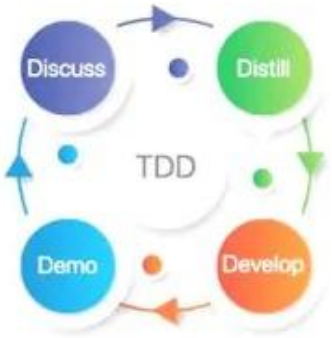
Comparative Analysis


Add Test
Refactor
Watch Test Fail
Run Test
Write Code

TDD


Scenarios
Behaviour

BDD


Discuss
Distill
TDD
Demo
Develop

ATDD