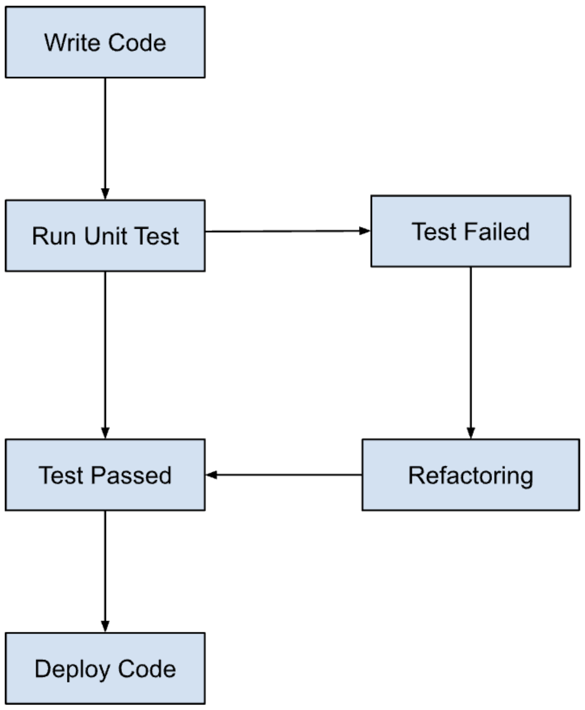
Assignment 1: Create an infographic illustrating the Test-Driven Development (TDD) process. Highlight steps like writing tests before code, benefits such as bug reduction, and how it fosters software reliability.

- **Approach**: TDD is an agile development methodology where tests are written before the code is developed. In contrast, traditional testing is performed after the code is written.
- **Testing Scope**: TDD focuses on testing small code units at a time, while traditional testing covers testing the system as a whole, including integration, functional, and acceptance testing.
- **Iterative**: TDD follows an iterative process, where small chunks of code are developed, tested, and refined until they pass all tests. The code is usually tested once and then refined based on the results in traditional testing.
- **Debugging**: TDD aims to catch errors as early as possible in the development process, making debugging and fixing them easier. Traditional testing, on the other hand, may require more effort to debug errors that are discovered later in the development process.
- **Documentation**: TDD documentation typically focuses on the test cases and their results, while traditional testing documentation may include more detailed information about the testing process, the test environment, and the system under test.
  Overall, TDD offers a more efficient and reliable approach to software development, ensuring that code is thoroughly tested before being integrated into the system. Traditional testing, however, may be more appropriate for larger and more complex projects where a more comprehensive approach to testing is required.

Three Phases of Test Driven Development

1. **Create precise tests:** Developers need to create exact unit tests to verify the functionality of specific features. They must ensure that the test compiles so that it can execute. In most cases, the test is bound to fail. This is a meaningful failure as developers create compact tests based on their assumptions of how the feature will behave.
2. **Correcting the Code:** Once a test fails, developers must make the minimal changes required to update the code to run successfully when re-executed.
3. **Refactor the Code:** Once the test runs successfully, check for redundancy or any possible code optimizations to enhance overall performance. Ensure that refactoring does not affect the external behavior of the program.

The image below represents a high-level TDD approach toward development:



Benefits of Test Driven Development Fosters the creation of optimized code.

1. It helps developers better analyze and understand client requirements and request clarity when not adequately defined.
2. Adding and testing new functionalities become much easier in the latter stages of development.
3. Test coverage under TDD is much higher compared to conventional development models. The TDD focuses on creating tests for each functionality right from the beginning.
4. It enhances the productivity of the developer and leads to the development of a codebase that is flexible and easy to maintain.

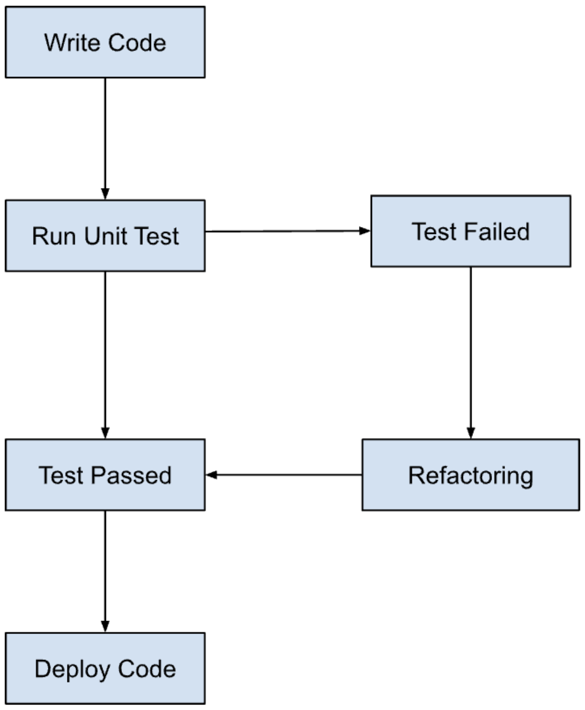
Assignment 2: Produce a comparative infographic of TDD, BDD, and FDD methodologies. Illustrate their unique approaches, benefits, and suitability for different software development contexts. Use visuals to enhance understanding

Test-Driven Development (TDD)

Approach: Write tests before writing code.

Benefits: Bug reduction, improved software reliability.

Suitability: Ideal for agile development, small to medium-sized projects.

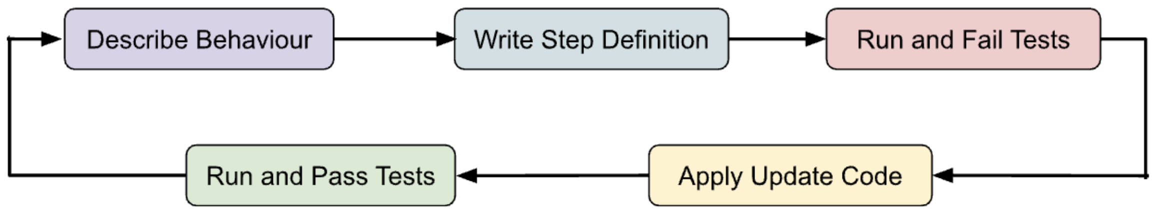


Behavior-Driven Development (BDD)

Approach: Focuses on behaviors and outcomes.

Benefits: Clear communication between stakeholders, encourages collaboration.

Suitability: Best for projects with complex business logic, large teams.



Feature-Driven Development (FDD)

Approach: Develop features incrementally.

Benefits: Scalability, emphasis on client needs.

Suitability: Suitable for large projects with well-defined requirements, waterfall methodology.

Conclusion:

Each methodology has its strengths and is suitable for different project contexts.

TDD emphasizes testing and bug prevention.

BDD focuses on collaboration and clear communication.

FDD prioritizes feature delivery and scalability.