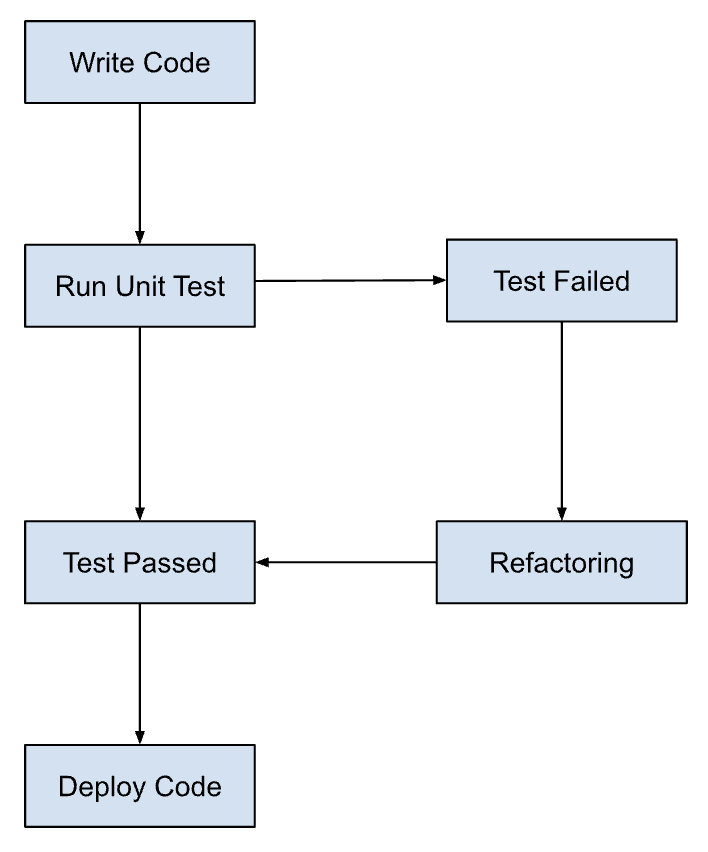
**Assignment1: Create an infographic illustrating the Test-Driven Development (TDD) process. Highlight steps like writing tests before code, benefits such as bug reduction, and how it fosters software reliability.**



1. **TDD Cycle Overview:**
   * Red, Green, Refactor Cycle:

* **Red**: Write a test for a new feature or improvement. Run the test and watch it fail. This ensures the test is valid and the feature isn't already present.
* **Green**: Write the minimal amount of code required to make the test pass. Focus on getting the test to pass rather than writing perfect code.
* **Refactor**: Improve the code's structure and readability without changing its behavior. Ensure all tests still pass

1. **Steps in TDD:**

* Write a test
* Identify a new feature or improvement.
* Write a test that defines the desired functionality.
* Run the test
* Run the test, which should fail since the code doesn’t exist yet.
* Write the code

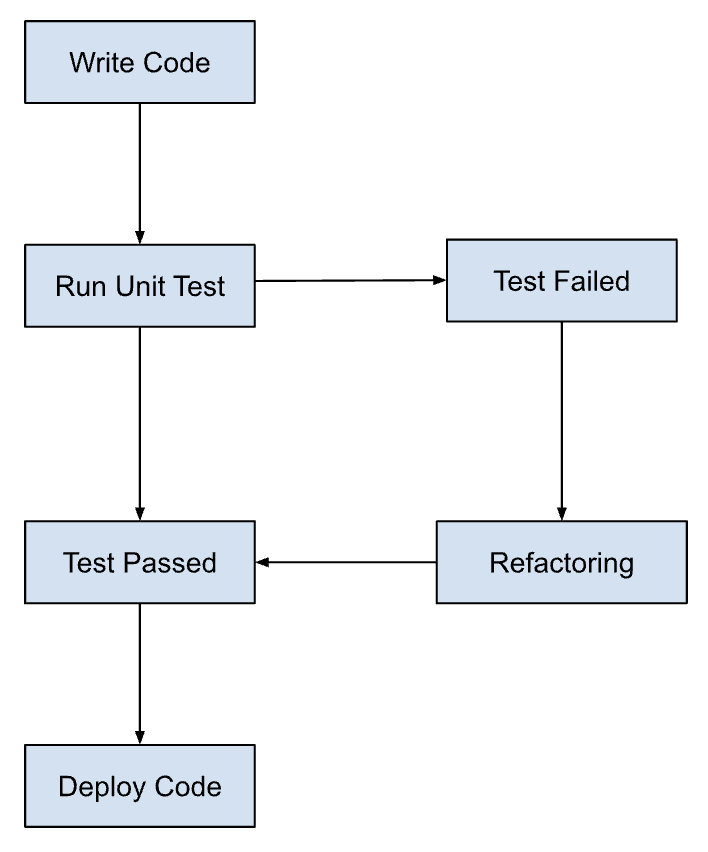
* Write the minimum amount of code required to pass the test.
* Run test Again
* Run all tests to ensure the new code doesn’t break anything.
* Refactor
* Clean up the code, improving it without changing the behavior.

Benefits of TDD:

* **Bug Reduction:**
* Bugs are caught early during the development process.
* Continuous testing ensures fewer bugs in the final product.
* **Software Reliability:**
* Ensures the code works as intended.
* Regular testing and refactoring lead to a more robust codebase.
* **Improved Design:**
* Encourages simpler, cleaner, and more modular code.
* Promotes better code design through regular refactoring.
* **Documentation:**
* Tests serve as documentation for the code base.
* Easy to understand the purpose and functionality of the code through the tests.
* **Confidence in Code Changes:**
* Facilitates fearless refactoring and code improvements.
* Developers can make changes with confidence that existing functionality remains intact.

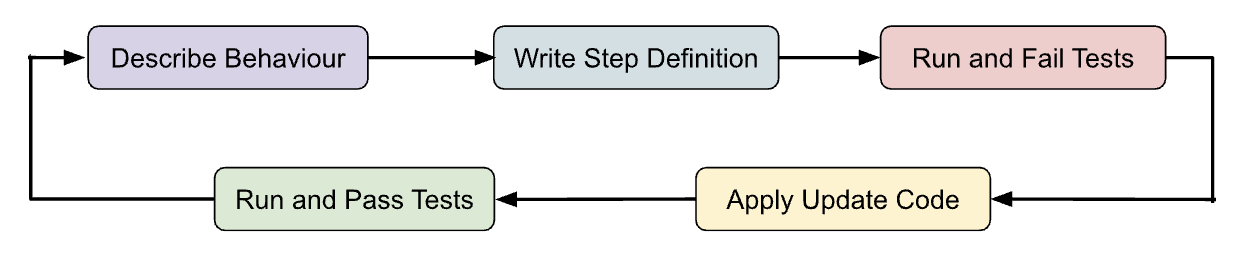
**Assignment 2: Produce a comparative infographic of TDD, BDD, and FDD methodologies. Illustrate their unique approaches, benefits, and suitability for different software development contexts. Use visuals to enhance understanding:**

1. TDD (Test-Driven Approach)



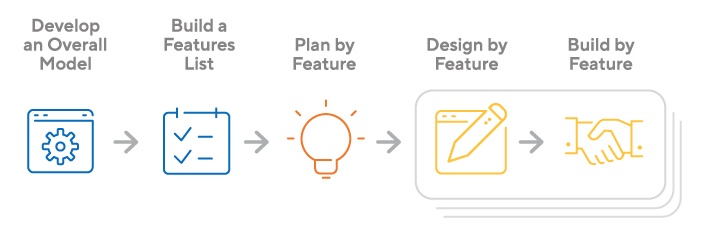
* **Approach:**
* Write tests before the code.
* Follow a cycle: Red (write a failing test) -> Green (write code to pass the test) -> Refactor (optimize the code)
* **Benefits:**
* Improved code quality and fewer bugs.
* Simplified debugging process.
* Promotes modular design.
* Tests serve as documentation.
* **Suitability:**
* Projects with clear and well-understood requirements.
* Teams with strong technical expertise.
* Development of low-level, technical components.

1. **BDD (Behavior-Driven Development):**



* **Approach:**
* Write tests in plain language (e.g., Gherkin) to describe behavior from a stakeholder’s perspective.
* Focus on scenarios: Given [context], When [event], Then [outcome].
* **Benefits:**
  + - Enhanced communication among developers, stakeholders, and non-technical teams.
    - Focus on delivering business value.
    - Reduced ambiguity in requirements.
    - Facilitates collaboration.
* **Suitability:**
* Projects with significant business involvement and complex requirements.
* Customer-facing applications.
* Teams with both technical and non-technical members.

1. FDD (Feature-Driven Development):



* **Approach:**
* Develop overall model and create a feature list.
* Plan, design, and build by feature, involving continuous iteration.
* **Benefits:**
* Client-driven development focusing on valuable features.
* Scalable for large teams.
* Encourages regular delivery of working software.
* **Suitability:**
* Large-scale projects with a need for scalable processes.
* Development environments requiring clear feature prioritization.
* Teams with strong leadership (Chief Programmers).

**Assignment 3: Write principles of Agile.**

Agile is a set of principles for software development under which requirements and solutions evolve through the collaborative effort of self-organizing cross-functional teams. Here are some key principles:

1. **Customer Satisfaction:**

* Deliver valuable software early and continuously.

1. **Welcome Change:**

* Embrace changes, even late in development.

1. **Deliver Frequently:**

* Release working software frequently, from a couple of weeks to a couple of months.

1. **Collaboration:**

* Business people and developers must work together daily.

1. **Motivated Individuals:**

* Build projects around motivated individuals and trust them to get the job done.

1. **Face-to-Face Conversation:**

* The most efficient and effective method of conveying information.

1. **Working Software:**

* The primary measure of progress.

1. **Sustainable Development:**

* Maintain a constant pace indefinitely.

1. **Excellence:**

* Continuous attention to technical excellence and good design.

1. **Simplicity:**

* Maximize the amount of work not done.

1. **Self-Organizing Teams:**

* The best architectures, requirements, and designs emerge from self-organizing teams.

1. **Reflection:**

* At regular intervals, the team reflects on how to become more effective.