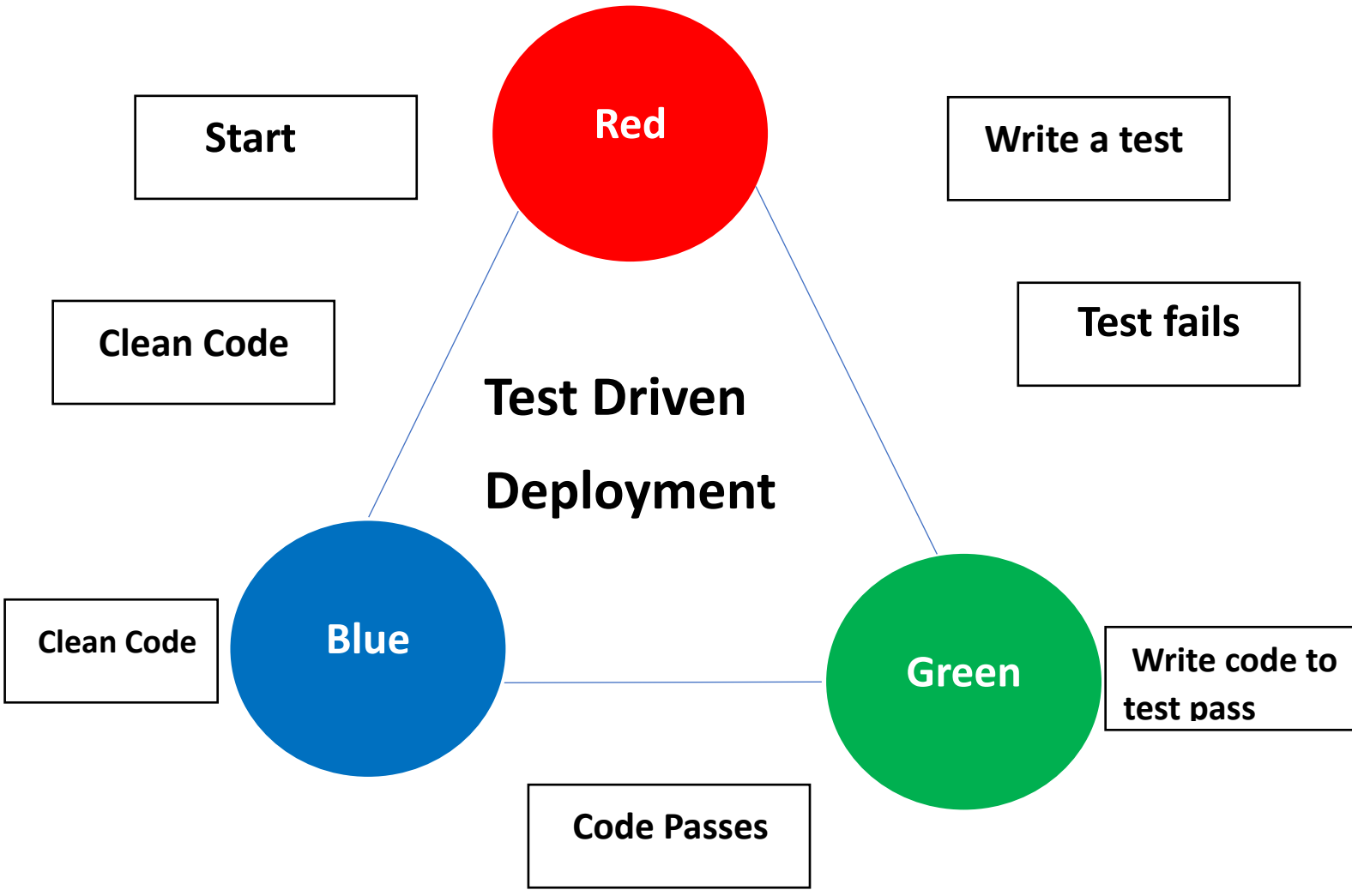
Assignment 01:

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.

Key 5 Steps to Implementing Test Driven Development



The key 5 steps of implementing test-driven development (TDD) are:

Step 1: Write a Test That Fails (Red Phase):

Start by identifying a specific thing you want your program to do or a new feature you want to add.

Create a test that describes what you expect the program to do, even though the code to make it happen doesn't exist yet.

The test should fail initially because the code hasn't been written to meet those expectations.

Step 2: Write the Simplest Code to Make the Test Pass (Green Phase):

Write the most basic code required to make the failing test pass. This usually means creating a function, defining variables, or writing fundamental logic.

Don’t worry too much about making the code perfect or efficient at this stage; the main goal is to get the test to work.

Step 3: Improve the Code (Refactor Phase):

After the test passes, take time to review and enhance the code. This could involve making it cleaner, faster, or easier to understand. Ensure that the code follows coding standards and best practices.

Run the tests again to confirm that your changes haven’t introduced new issues. All tests should still pass.

Step 4: Run All Tests (Green Phase Again):

Before moving on to new work or changes, run all the tests, including the existing ones, to make sure your modifications haven’t broken any previously working parts of the program.

If any tests fail, fix the problems immediately and ensure that all tests are passing again before proceeding.

Step 5: Repeat the Process (Red-Green-Refactor):

Go back to Step 1 and repeat the cycle for the next thing you want your program to do or the next feature you want to add.

Write a test first, create the code to make it work, and then improve the code as needed.

Continue this iterative approach until you've implemented all the desired functionality or features.

TDD encourages a systematic approach to programming, where you ensure your code works correctly from the very beginning, making it easier to maintain and less prone to errors as you build upon it.