* **Test Driven Development (TDD)**

**Test Driven Development (TDD)** is a software development practice where developers write automated tests before writing the actual code that needs to be tested. Developers create unit test cases before developing the actual code. It is an iterative approach combining Programming, Unit Test Creation, and Refactoring.

The process follows a repetitive cycle known as **Red-Green-Refactor.**

1. **Red Phase**: First, a developer writes a test that defines a desired feature or behavior (the “Red” phase, as the test will fail initially).
2. **Green Phase**: Then, they write the minimum code necessary to pass the test (the “Green” phase).
3. **Refactor**: Finally, the code is refactored for optimization while ensuring the test still passes.

TDD helps ensure that the codebase remains reliable and bug-free by catching errors early in the development process. It promotes better design decisions, as writing tests first forces developers to think more clearly about the functionality they are implementing.

Furthermore, because tests are an integral part of the development process, TDD leads to higher code coverage and makes future modifications or refactoring easier and safer, knowing that existing functionality is thoroughly tested.

* The TDD approach originates from the Agile manifesto principles and Extreme programming.
* As the name suggests, the test process drives software development.
* Moreover, it’s a structuring practice that enables developers and testers to obtain optimized code that proves resilient in the long term.
* In TDD, developers create small test cases for every feature based on their initial understanding. The primary intention of this technique is to modify or write new code only if the tests fail. This prevents duplication of test scripts.

**Test Driven Development (TDD) Examples**

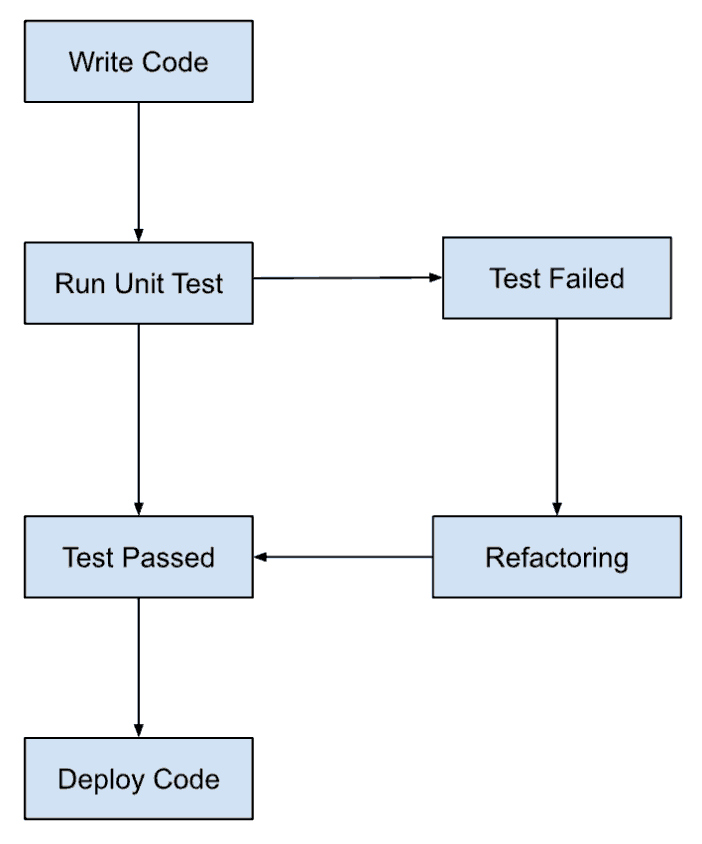
Here are some of the examples where TDD is used:

1. **Calculator Function**: When building a calculator function, a TDD approach would involve writing a test case for the “add” function and then writing the code for the process to pass that test. Once the “add” function is working correctly, additional test cases would be written for other functions such as “subtract”, “multiply” and “divide”.
2. **User Authentication**: When building a user authentication system, a TDD approach would involve writing a test case for the user login functionality and then writing the code for the login process to pass that test. Once the login functionality works correctly, additional test cases will be written for registration, password reset, and account verification.
3. **E-commerce Website**: When building an e-commerce website, a TDD approach would involve writing test cases for various features such as product listings, shopping cart functionality, and checkout process. Tests would be written to ensure the system works correctly at each process stage, from adding items to the cart to completing the purchase.

**Three Phases of Test Driven Development**

1. **Create precise tests:** Developers need to create exact [unit tests](https://www.browserstack.com/guide/unit-testing-a-detailed-guide) 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 (TDD)**

1. Fosters the creation of optimized code.
2. It helps developers better analyze and understand client requirements and request clarity when not adequately defined.
3. Adding and testing new functionalities become much easier in the latter stages of development.
4. 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.
5. It enhances the productivity of the developer and leads to the development of a codebase that is flexible and easy to maintain.