**Test-Driven Development**

* Can create code that is more fault-tolerant
* Write tests first, then write code. Implement changes until your code passes the tests you wrote.
* Unit tests are simple, cover small portions of logic (like an algorithm).
* Sometimes referred to as *Test First Development*
* Use mock data for unit tests rather than random/changing data (like API calls)

**5 steps in TDD flow:**

1. Read, understand, and process the feature or bug request.
2. Translate the requirement by writing a unit test. If you have hot reloading set up, the unit test will run and fail as no code is implemented yet.
3. Write and implement the code that fulfills the requirement. Run all tests and they should pass, if not repeat this step.
4. Clean up your code by refactoring.
5. Rinse, lather and repeat.

**Source:** [5 steps of test-driven development - IBM Develope](https://developer.ibm.com/articles/5-steps-of-test-driven-development/)r

Unlike unit testing that focuses only on testing the functions, classes, and procedures, TDD drives the complete development of the application. Therefore, you can also write functional and acceptance tests first. **Source:** [Test-driven development - IBM Garage Practices](https://www.ibm.com/garage/method/practices/code/practice_test_driven_development/)

<https://www.codica.com/blog/test-driven-development-benefits/>

TDD is time and cost-consuming only temporarily. It can have advantages in the long run. Write a functional test before coding is done. Was based on a method original conceive in the early days of programming with tape as the input and output. Older programmers know this as the main way to program.

1. Write a test that fails because of code absence

2. Write code that solves the test.

3. Refactor the code and improve it without changing the external behavior.

Repeat until full functionality is achieved.

Avoids repeating code in different system parts. Encourages small classes and functions and keep clearly defined system objects. Build code to TDD usage and improve quality. Code can become simple, straightforward, and brief. Detailed documentation can be made, and less time can be required for development.

<https://www.ibm.com/garage/method/practices/code/practice_test_driven_development/>

Code and tests are constructed at the same time to build up to a good result, rather than making a good one from the start. Code is written to a test case. Three Laws of TDD:

* You are not allowed to write any production code unless it is to make a failing unit test pass.
* You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures.
* You are not allowed to write any more production code than is sufficient to pass the one failing unit test.

Two-rule version:

* Write only enough of a unit test to fail.
* Write only enough production code to make the failing unit test pass.

Makes code flexible and extendable. Tests are tested; they are designed to fail. Only the function that is needed is written. Start by writing a test that confirms a behavior, Add only enough to make a new test case and pass the others. Refactor and eliminate duplicate code.

Ensure the methodology is respected by project leadership. Writing tests can take more effort. A failing test must stop the pipeline.

**Red phase** – write an automated test. Consider the requirements for a user and it’s appearance.

**Green phase** – Write code, but only enough to pass. Do not think about how to make it pretty or if it follows best practices. Only consider making it work. It makes it less prone to errors.

**Refactor phase** – Change the code so it becomes better. Remove duplication and make it more general. Reuse the test to make sure functionality is maintained. Tests should not need to be chagned.

IBM finds that built-in testing produces better code. TDD coders produced better code than teams who used other techniques.

<https://www.techtarget.com/searchsoftwarequality/definition/test-driven-development>

Also called test-driven design. Introduced with Extreme Programming(XP). Started by creating a failing unit test. Code just enough to fix the problem. After passing, change the design without changing behavior. Focuses more on the programmer with the project. Collaboration with customers and technical staff is important to make useful tests. Applications of high quality can be produce in less time than older methods. Requires anticipation of real-world use. Regression testing can be done to automate testing. Debugging at later stages can be lessened. Requires considerable skill and clean code. Tests need to be updated and maintained. Final results are only as good as the tests conducted.

**Example of Test-Driven Development (TDD):**

[InfoQ](https://www.infoq.com/articles/test-driven-design-java/)