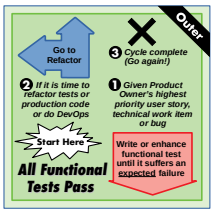
TDD Training Cube — TDD Outer Loop

TDD Learnstitute



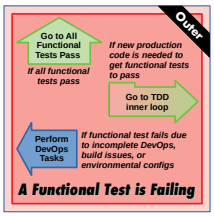
Always start a new software development cycle with the **All Functional Tests Pass** side facing up. Run functional tests at start to ensure all functional tests are passing.

The activities on this side are numbered in rank order and are optional. On each visit to this side of the cube, perform one of the listed activities that would be best for achieving your Product Owner's current objective while also producing the highest quality code.

If you have your Product Owner's highest priority user story, technical work item or bug - ❶
The overall goal of the TDD Outer Loop is to let prioritized user stories articulated by a Product Owner drive software development. Given the highest priority user story, technical work item or bug, write or enhance a related functional test that expresses the feature to be developed. Edit and run your new functional test until it fails in a way you expect. Then, turn to the **A Functional Test is Failing** side.

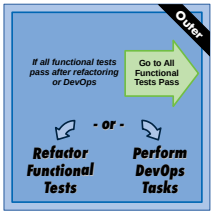
If you see an opportunity to refactor or if you need to do DevOps - ❷
As long as all tests are passing, you may refactor any parts of your software or do DevOps as needed. Turn to **Refactor Functional Tests**, **Perform DevOps Tasks**, **Refactor Unit Tests**, or **Refactor Production Code**.

Otherwise, if all functional tests are passing - ❸
If you have completed the highest priority work item from your Product Owner, then this software development cycle is complete! Start a new cycle or quit for the day.



This side faces up whenever you discover that **A Functional Test is Failing** or have just returned from the TDD Inner Loop. This side is a decision point. Immediately turn the cube based on the conditions listed.

If a functional test is failing due to missing production code, turn to the TDD Inner Loop (starting with the **All Unit Tests Pass** side) and write code that will contribute to getting the functional test to pass. If a functional test is failing because DevOps is required, turn to the **Perform DevOps Tasks** side. If all tests are passing, turn to the **All Functional Tests Pass** side.



This side represents TDD Outer Loop refactoring activities. When you turn to this side, you may either **Refactor Functional Tests** or **Perform DevOps Tasks**, but never both at the same time. Take care to preserve the semantics of your tests while refactoring.

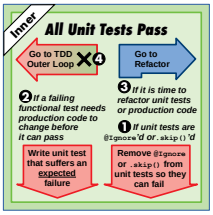
After refactoring or DevOps, run your functional tests. If all functional tests pass, turn to the **All Functional Tests Pass** side.

Glossary of Terms

System under test (SUT) is the subject of a software test. The SUT is a subset of your software that is tested in isolation from the rest of your software. Isolating the SUT can bring beneficial decoupling effects to your design.
Functional tests are tests which exercise your production code from the perspective of your users. Also known as "outside" or "black box" tests, functional tests might call a library from its exported API, call a service's external REST endpoints, or interact with a user experience via a web driver just like a real user would. The SUT for a functional test is typically a single external interface exposed on a running application.
Unit tests are tests which exercise a single module of a software system. Also known as "inside" tests, the SUT for unit tests are typically the smallest parts of a software system, like a class, function, or component.
Production code is any tested code that ships with your software. Shipped, but untested code is **legacy code**.
DevOps are any non-coding tasks needed to test or deploy your software. This includes editing build scripts or maintaining build servers, configuring deployment servers, data source setup, and logging/metrics analysis.

TDD Training Cube — TDD Inner Loop

TDD Learnstitute



The overall goal for the TDD Inner Loop is to write production code in support of the currently failing functional test. The activities on this side of the cube are numbered in rank order and are optional. On each visit to this side of the cube, perform one of the listed activities that would make progress in getting the failing functional test to pass while also producing the highest quality code.

If you have tests you previously marked `@Ignore` or `.skip()` in order to transform production code - ❶
Remove the `@Ignore` or `.skip()` from the tests. Run unit tests. If any of the unignored tests fail, turn to **A Unit Test is Failing**.

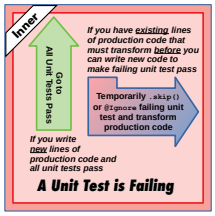
---

*If you turned to **All Unit Tests Pass** from **A Functional Test is Failing*** - ❷
Discover where in your production code that changes are needed. Sometimes, the failing functional test's error logs will hint at which module in your production code needs to be modified. Otherwise, search through the production code starting at the entry point the failing functional test uses to enter the program. Trace through the layers of the production code, stopping at the first module where modifications would make progress towards getting the functional test to pass.

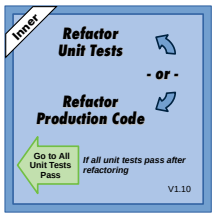
After discovering which module you wish to write new production code in, write the simplest unit test for that module that you can devise that would advance your production code towards passing the failing functional test. Run the unit tests. Once your new test gets an expected failure, turn to **A Unit Test is Failing**.

If you see opportunities to refactor - ❸
Turn to the **Refactor Unit Tests** or **Refactor Production Code** side.

Otherwise, if all unit tests are passing - ❹
Run your functional tests and then turn to the **A Functional Test is Failing** side. Turn regardless of whether the functional tests now pass or fail.



The code you will change to get your unit test passing will typically be in a single function in a production module. This may require changing existing lines of code. These changes to existing code are called *transformations*. (See the *Transformation Priority Premise* wiki for discussion and links - `https://goo.gl/HCEBoZ`) If you must apply the transformations to existing lines of code, do so by refactoring. First, mark the failing unit test with `@Ignore` or `.skip()`, (as appropriate for the test framework.) All remaining unit tests should pass. Next, turn to the **Refactor Production Code** side. Apply the transformation such that all remaining unit tests continue to pass. Otherwise, if you are only adding new lines to the production code, add them to make the failing unit test pass. Then, turn to the **All Unit Tests Pass** side.



This side represents TDD Inner Loop refactoring activities. When you turn to this side, you may either **Refactor Unit Tests** or **Refactor Production Code**, but never both at the same time. Take care to preserve the semantics of the tests and production code while refactoring.

After refactoring, run the unit tests. When all tests pass, turn to **All Unit Tests Pass** side. To learn more about refactoring, visit `https://refactoring.com`

Outside-In TDD Looping Tips

• Functional tests are rarely comprehensive upon first writing due to lack of design knowledge about what should live in production code. Try to write the simplest functional test you can that will advance the user story, turn to the TDD Inner Loop to get that passing, and return to the TDD Outer Loop for functional test elaboration. Repeat.
• A visit to the TDD Inner Loop can be very short, on the order of a minute or three. Limit yourself to a writing a single unit test per visit, get that unit test to pass by modifying production code, and return to the TDD Outer Loop to see what progress was made in getting the functional test to pass. Let functional test failures drive the process.