Agile Software Development (TCS 855)

Unit-III Agile Software Design and PROGRAMMING

Test Driven Development (TDD)

Prof.(Dr.) Santosh Kumar

Department of Computer Science and Engineering

Graphic Era Deemed to be University, Dehradun

Test Driven Development (TDD)

- **Test Driven Development** is the process in which test cases are written before the code that validates those cases. It depends on repetition of a very short development cycle. Test driven Development is a technique in which automated Unit test are used to drive the design and free decoupling of dependencies.
- The following sequence of steps is generally followed:

i. Add a test – Write a test case that describe the function completely. In order to make the test cases the developer must understand the features and requirements using user stories and use cases.

ii. Run all the test cases and make sure that the new test case fails.

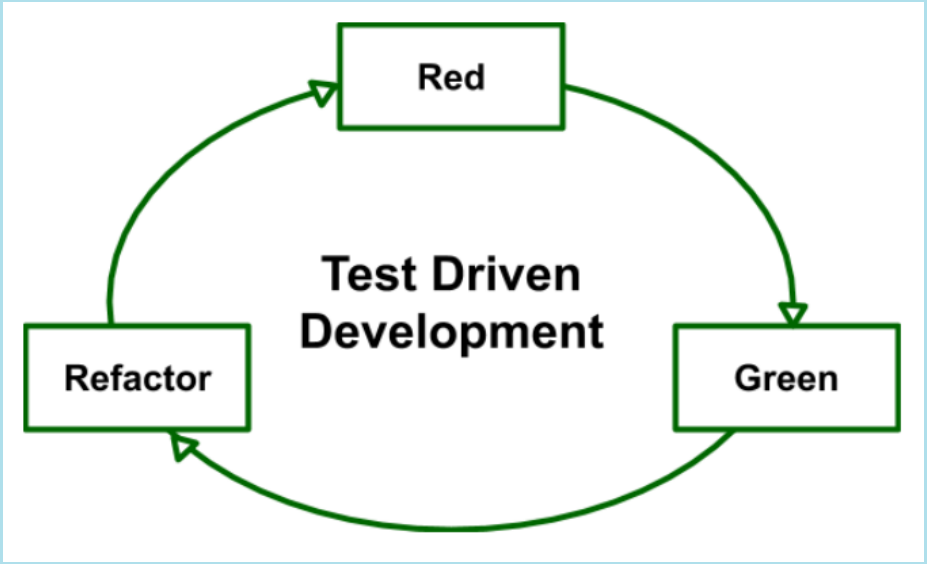
iii. Write the code that passes the test case

iv. Run the test cases

v. Refactor code – This is done to remove duplication of code.

vi. Repeat the above mentioned steps again and again

Motto of TDD



i. **Red –** Create a test case and make it fail
ii. **Green –** Make the test case pass by any means.
iii. **Refactor –** Change the code to remove duplicate/redundancy.

Benefits:

- Unit test provides constant feedback about the functions.
- Quality of design increases which further helps in proper maintenance.
- Test driven development act as a safety net against the bugs.
- TDD ensures that your application actually meets requirements defined for it.
- TDD have very short development lifecycle.

Advantages and disadvantages of Test Driven Development (TDD)

Test Driven Development(TDD)

- Test-Driven Development (TDD) is additionally called test-driven design. TDD may be a method of software development during which ASCII text file is tested over and once again (unit testing).
- Test-driven development may be a balanced approach for the programming perfectly blended with tightly interwoven three activities: coding, testing (writing unit tests), and designing (refactoring)first goal of correcting specification instead of the validation first.
- In other words, TDD may be a smart approach to know and streamline the wants before writing the functional code within the line of Agile principles.

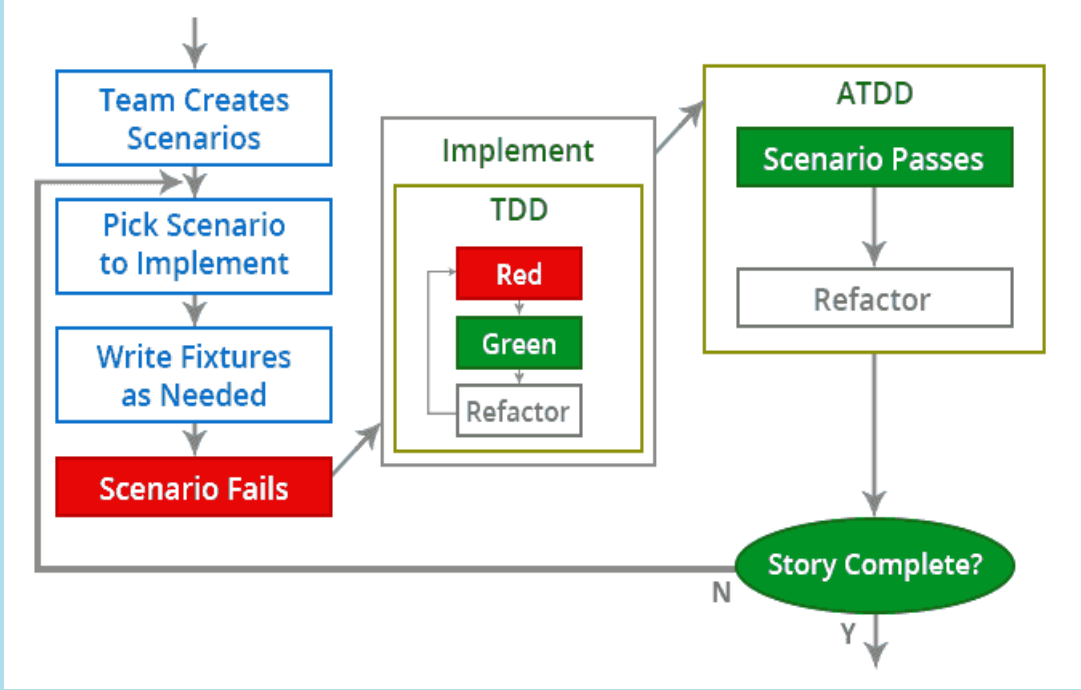
Advantages of TDD

- **You only write code that's needed –**
  Following the principles, you've got to prevent writing production code when all of your tests pass. If your project needs another feature, you would like a test to drive the implementation of the feature. The code you write is the simplest code possible. So, all the code ending up within the product is really needed to implement the features.
- **More modular design –**
  In TDD, you consider one microfeature at a time. And as you write the test first, the code automatically becomes easy to check. Code that's easy to check features a clear interface. This leads to a modular design for your application.
- **Easier to maintain –**
  Because the different parts of your application are decoupled from one another and have clear interfaces, the code becomes easier to take care of, you'll exchange the implementation of a microfeature with a far better implementation without affecting another module. you'll even keep the tests and rewrite the entire application. When all the tests pass, you're done.
- **Easier to refactor –**
  Every feature is thoroughly tested. you do not get to be afraid to form drastic changes because if all the tests still pass, everything is ok. Now, is extremely important because you, as a developer, improve your skills each and each day. If you open the project after six months of performing on something else, most likely, you will have many ideas on the way to improve the code. But your memory about all the various parts and the way they fit together isn't fresh anymore. So, making changes is often dangerous. With an entire test suite, you'll easily improve the code without the fear of breaking your application.
- **High test coverage –**
  There's a test for each feature. This leads to a high test coverage It develops gain confidence in your code.
- **Tests document the code –**
  The test code shows you ways your code is supposed to be used. As such, it documents your code. The test code is a sample code that shows what the code does and the way the interface has got to be used.
- **Less debugging –**
  How often have you ever wasted each day to seek out a nasty bug? How often have you copied a mistake message from Xcode and looked for it on the web.

Disadvantages of TDD

- **No silver bullet –**
  Tests help to seek out bugs, but they can not find bugs that you simply introduce within the test code and in implementation code. If you haven't understood the matter you would like to unravel, writing tests most likely doesn't help.
- **slow process –**
  If you begin TDD, you'll get the sensation that you simply need an extended duration of your time for straightforward implementations. you would like to believe the interfaces, write the test code, and run the tests before you'll finally start writing the code.
- **All the members of a team got to do it –**
  As TDD influences the planning of code, it's recommended that either all the members of a team use TDD or nobody in the least. additionally, to the present, it's sometimes difficult to justify TDD to the management because they often have the sensation that the implementation of latest features takes longer if developers write code that will not find themselves within the product half the time. It helps if the entire team agrees on the importance of unit tests.
- **Tests got to be maintained when requirements change –**
  Probably, the strongest argument against TDD is that the tests need to be maintained because the code has got to. Whenever requirements change, you would like to vary the code and tests. But you're working with TDD. this suggests that you simply got to change the tests first then make the tests pass. So, actually, this disadvantage is that the same as before when writing code that apparently takes an extended time.y takes a long time.

Test Driven Development Workflow



- Test Driven Development promotes the idea of each test case testing one piece of functionality at a time. The workflow of Test Driven Development is as follows –Write a concise, minimal test for a new piece of functionality. This test will fail since the functionality isn't implemented yet (if it passes, one knows that the functionality either already exists or that the test isn't designed correctly).
- Implement the new functionality and run all tests, both the new and pre-existing ones. Repeat until all tests pass.
- Clean up the code and ensure that all tests still pass, then return to step 1.

Why Test Driven Development Matters?

- Requirements - Drive out requirement issues early (more focus on requirements in depth).
- Rapid Feedback - Many small changes Vs. One significant change.
- Values Refactoring - Refactor often to lower impact and risk.
- Design to Test - Testing driving good design practice.
- Tests as information - Documenting decisions and assumptions.

>Test Driven Development helps the programmer in several ways, such as -

- Improve the code.
- Side by side, increasing the programmer’s productivity.

>Using Test Driven Development concept in one's programming skills -

- It will save the developer's time which is wasting rework.
- Able to identify the error/problem quicker and faster.
- The programmer will write small classes focused only on a single functionality instead of writing the big classes.
- Whenever the code base gets more prominent, it becomes tough to change and debug the code. Moreover, there is a high chance of mess up of the code.

>But, if developers are using Test Driven Development technique -

- Means developers have automated tests.
- Writing the test cases for the program is a safe side for the programmers.
- It becomes easy to view what the error is, where it is and how it is paralyzing one's code.

Need For Test Driven Development

- Ensures quality – It helps in ensuring the quality by focusing on requirements before writing the code.
- Keeps code neat and tidy – It assists in keeping the code clear, simple, and testable by breaking it down into small achievable steps.
- Maintains Documentation – It provides documentation about how the system works for anyone coming into the team later.
- Repeatable Test and rapid change – It builds a suite of repeatable regression tests and acts as an enabler for rapid change.

Best Practices to Adopt Test Driven Development

- The best practises adopting Test Driven Development are below.

Road Map of TDD

- One of the best practices is to clear out with thought and further break it into the test case. Follow the red-green approach to build the test case.
- The first step is to create the red test, and after exposing all the problems related to code, make some changes and make it a green test.

Implementation of TDD

- It is essential to implement both source code and test case separately. For both implementation and testing, there should be two directories. In every programming language, there should be different packages for both.
- For example, in Java, “src/main/java” is beneficial for implementation, and on the other hand, “src/test/java” helps in testing.

Structure of Test Driven Development

- The structure for writing test cases should be correct. It is common practice to write the test class with the same name used in the production/implementation class, but there should be a change in the suffix.
- Consider an example; if the implementation/production class is “Student,” then the test class should be “StudentTest.” And similarly, in the case of methods, test methods are written with the same name as production methods, but there should be a change in the prefix, like if the method name is “display student name,” then in testing, it should be “testDisplayStudentName.”

Test Driven Development with Scala

- Scala is a programming language that combines both object-oriented and functional programming paradigms. Considering functional aspects of Scala, we can define functions of two types –Functions that perform computation and no I/O operations –
  - In general, the developer must focus on writing functions that are free of side effects.
  - If the function does have side effects, those should not affect the results of subsequent tests.
- Functions that perform I/O operations and no computation –
  - If a function performs less computation and more IO operations, one must consider specific testing frameworks for writing tests for such functions. For example, there is no need to use ScalaTest (a popular testing framework) for such functions.
- In scala, tests reside in the project source but are separate from the application code. There are many testing frameworks available with a lot of features. One can choose according to the specific features in the framework. We will consider ScalaTest, a popular testing framework, in this document.