**Test Driven Development**

Date: 06/08/2018

Test Driven Development is an evolutionary process where we first write the test before writing the actual code. The primary goal of this approach is not only validation but understanding the requirements and design before we implement the functional code.

**TDD:**

TDD involves set of functional steps. The first step is to add a test case (which is enough to fail a code). Then we run our testcase. As it fails, now updating the functional code should happen to pass the testcases. Now make new tests. The fourth is step is to run tests again. When they get failed code should be updated again. Once a feature is written properly, then we’ll be moving t next feature.

To be a bit mathematical we can say TDD as

TDD = Refactoring + TFD

**TFD :** Test First Development

Whenever we take up a task, we first check whether the design is the best design possible or not. If so proceed via TFD approach. If not, refactor the code locally to change the part of design affected by the new feature. A significant advantage of TDD is that it enables you to take small steps when writing software. This is a practice that is promoted for years because it is far more productive than attempting to code in large steps. By this approach bugs can identified easily and fixed soon.

**TDD Cycle:**

1. First step involves understanding domain to get started.
2. Initial design should be made.
3. A test case should be written quickly, and we should run it.
4. If it succeeds, run all the tests, Else write the code.
5. Once all the testcases are executed empty the TODO List
6. If all tests are not passed, clean the code quickly.
7. If testcases are passed, start over next feature.

**TDD process smells:**

1. If there is no bar in last 10 minutes, make the steps smaller and get feedback as soon as possible
2. Writing production code before running test.
3. Not spending enough time on refactoring.
4. Skipping something which is too easy to test.
5. Skipping something too hard to test.
6. Organizing tests around methods, not behavior
7. Using code coverage as goal.

**Green bar patterns:**

1. On step Test : Picking up a test which we are confident enough to implement and maximize learning effect.
2. If we remember about new test, write that on the TODO list and work on the present test.
3. Write tests on external components to check their behavior.
4. First check whether the test is running or not. Refactor that later.
5. A test should be written at least with two sets of sample data.
6. Implementation to single element should be done first. Then we can extend that to several elements.

**Junit:**

Junit is a unit testing framework for Java. Junit is a tool for Test Driven Development. Junit comes up the idea of “testing first, coding next”, which is the principle of test driven development. This increases the productivity and stability of program code.

**Features of Junit:**

1. JUnit is an open source framework, which is used to test the source code.
2. Annotations to identify test methods.
3. Assertions are to test expected results.
4. Provides test runners to run tests.
5. JUnit helps to write code faster with an increased quality.
6. JUnit tests can be organized into test suites containing testcases.

Junit framework is a package of classes which allows us to write tests for each method. Test Runner runs the tests and reports Test Results. One can test a class by extending an abstract class TestCase.

**Assert Class:**

Assert class have assert methods which are useful to write only tests.

Assert Methods:

1. void assertEquals(Boolean excepted, Boolean actual)
2. void assertTrue(Boolean condition)
3. void assertFalse(Boolean condition)
4. void assertNotNull(Object object)
5. void assertNull(Object object)
6. void assertSame(object1,object2)
7. void assertNotSame(object1,object2)
8. void assertArrayEquals(expectedArray,resultant array)

**Annotations:**

Annotations are like meta tags which provide information about test methods.The information can be like:

* which methods are going to run before and after test methods.
* Which methods will run before and after all test methods.
* Which methods should be ignored during execution.

Annotations:

1. @Test: This annotation tells JVM that it can be treated as a testcase.
2. @Before: This annotation makes a method to be run before each Test method.
3. @After: Annotating a public void method with @After causes that method to be run after the Test method.
4. @BeforeClass: It causes a method to run before every test method in the class.
5. @AfterClass: This is performed once after all the test methods are executed. Generally used to perform clean up activities.
6. @Ignore: When this annotation is attached to a public void method, JVM ignores that particular method.