Research the aspects of Test Driven Development

Take some time and investigate the aspects of Test Driven Development. What kind of tests are there? What are some good rules to creating tests? How do you determine a good test.

Test-driven development (TDD) is a approach in software development process that relies on the repetition of a very short development cycle: first the developer writes an (initially failing) automated test case that defines a desired improvement or new function, then produces the minimum amount of code to pass that test, and finally refactors the new code to acceptable standards.

TDD is a thought process of thinking testing from different perspective. In a normal scenario, we first write the code and then write the test case. In TDD we first write the test case before writing a single line of code. It is also called RED, GREEN, REFACTORING process.

The following sequence of steps is generally followed:

Add a test
Run all the tests and see if the new test fails
Write the code
Run tests
Refactor code
Repeat

There are various aspects to using test-driven development, for example the principles of "keep it simple, stupid" (KISS) and "You aren't gonna need it" (YAGNI). By focusing on writing only the code necessary to pass tests, designs can often be cleaner and clearer than is achieved by other methods. In Test-Driven Development by Example, Kent Beck also suggests the principle "Fake it till you make it".

Unlike the traditional software process Design - Implement - Test, in TDD it is Test/Design - Implement. The phases are much shorter than the original phases. Instead of a phase taking several months they take several minutes. We work in small steps which are all so simple that we can't get them wrong. So Test and Design are one phase, we design the system by writing tests which will use the system. This helps us to design a system easy to use and easy to test, a byproduct of this design activity is the

automated unit tests which can be easily run to prove that the system still works. This means we can easily and quickly do the regression test of the whole codebase and verify that we haven't broken anything that used to work. Not only that if we do break something we can fix it or revert it quickly which is the cheapest time to fix the bug.

TDD is the practice of writing an automated unit test to test your code before you write the code. The process of TDD starts -

By thinking about what you want to write
Write a test that would prove that you have written what you intended to write
Implement so that test passes
Run all of the unit tests and check that they all still pass
Refactor the new code you wrote to make it easier to maintain
Run all of the units again to check that they all still pass
Check your code into source control.

TDD Tools

TDD can be applied to any language and IDE like C#, VB.Net, Java, Ruby, Eclipse, Visual Studio. There are xUnit frameworks for most languages like JUnit for Java, NUnit for C#, cppUnit for C++, there are frameworks for javascript PHP and python.

In TDD, there is different kinds of testing:

Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation. Unit testing can be done manually but is often automated.

Regression testing is the process of testing changes to computer programs to make sure that the older programming still works with the new changes. Regression testing is a normal part of the program development process and, in larger companies, is done by code testing specialists.

Term “End to End testing” is defined as a testing method which determines whether the performance of an application is as per the requirement or not. It is performed from start to finish under real-world scenarios like communication of the application with hardware, network, database and other applications.

The main reason for carrying out this testing is to determine various dependencies of an application as well as ensuring that accurate information is communicated between various system components. It is usually performed after completion of functional and system testing of any application.

How to determine a good test -

There are three basic elements to look if it is a good test — reliability, validity, and standardization.

RELIABILITY is a measure of the test's consistency. A useful test is consistent over time. As an analogy, think of a bathroom scale. If it gives you one weight the first time you step on it, and a different weight when you step on it a moment later, it is not reliable. Reliability also can be a measure of a test's internal consistency. All of the items (questions) on a test should be measuring the same thing — from a statistical standpoint, the items should correlate with each other. Good tests have reliability coefficients which range from a low of .65 to above .90 (the theoretical maximum is 1.00).

VALIDITY is a measure of a test's usefulness. Scores on the test should be related to some other behavior, reflective of personality, ability, or interest. For instance, a person who scores high on an IQ test would be expected to do well in school or on jobs requiring intelligence.

STANDARDIZATION is the process of trying out the test on a group of people to see the scores which are typically obtained. In this way, any test taker can make sense of his or her score by comparing it to typical scores. This standardization provides a mean (average) and standard deviation (spread) relative to a certain group. When an individual takes the test, she can determine how far above or below the average her score is, relative to the normative group. When evaluating a test, it is very important to determine how the normative group was selected. For instance, if everyone in the normative group took the test by logging into a website, you are probably being compared to a group which is very different from the general population.

****