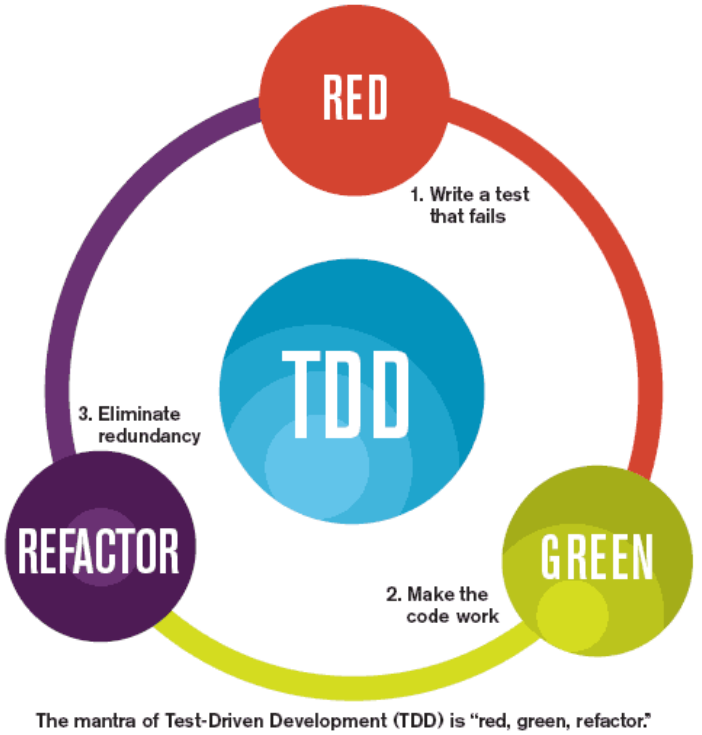
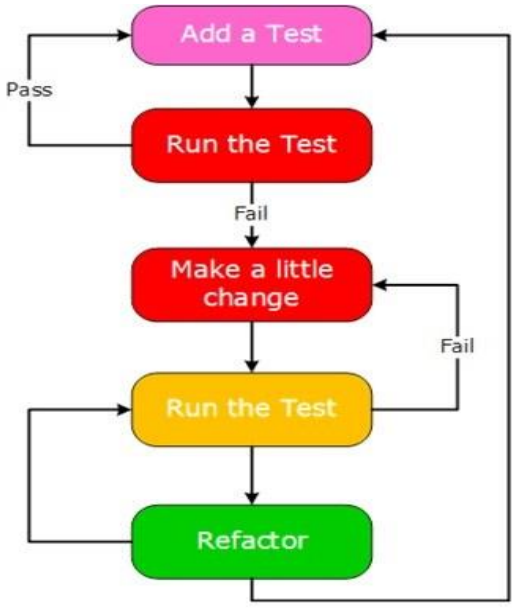
TDD (Test-Driven Development)

TDD is a development process that relay on very short repetition development cycle. Requirements are turned into very specific test case. The development have to pass those new test.



Life Cycle: Add Test-> Run All Test and see if the new tests fails-> write code -> run tests -> Refactor Code -> Repeat;



TDD (Test-Driven Development)

There are various kind of test. I am explaining 3 of them:

- Unit Test
- Integration Test
- End-End Test

Unit Test: Unit testing is a software development process in which the smallest testable parts of an application, called Units. Those test doesn't depend on other environment like text file, JSON, xml, word, excel or database;

Integration Test: Those tests are depends on other unit test and other environment integration like database, text files etc.

End- end test: Those tests a full function test like it have to sign in, go to the form and give all data form and then it verify. All those test should do by code. This actually like a manually test by code. This kind of test are not preferable now a days.

Benefits of TDD

- **Early bug notification.**

  Developers test their code but in the database world, this often consists of manual tests or one-off scripts. Using TDD you build up, over time, a suite of automated tests that you and any other developer can rerun at will.

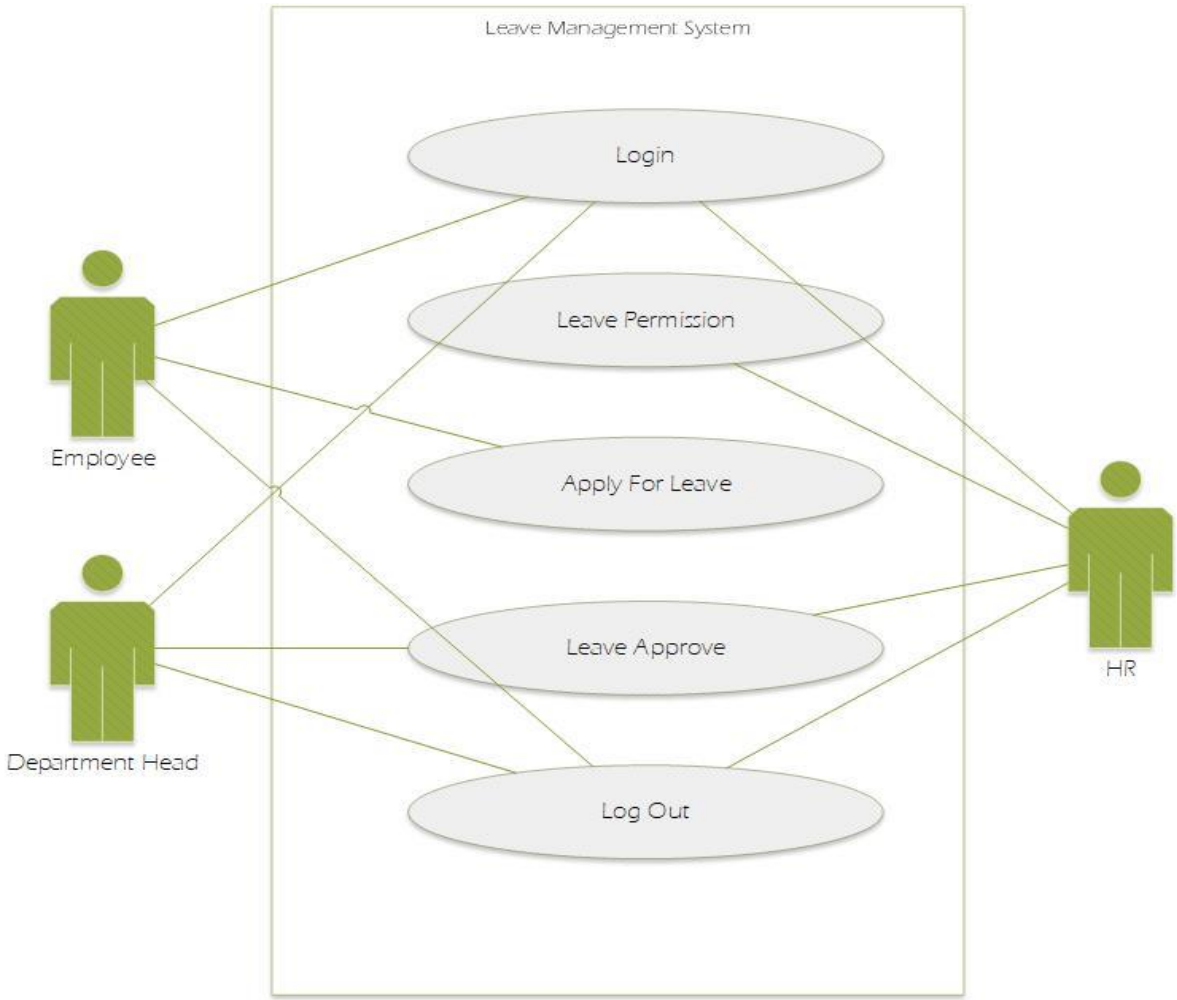
- **Better Designed, cleaner and more extensible code.**
  - It helps to understand how the code will be used and how it interacts with other modules.
  - It results in better design decision and more maintainable code.
  - TDD allows writing smaller code having single responsibility rather than monolithic procedures with multiple responsibilities. This makes the code simpler to understand.
  - TDD also forces to write only production code to pass tests based on user requirements.
- **Confidence to Refactor**
  - If you refactor code, there can be possibilities of breaks in the code. So having a set of automated tests you can fix those breaks before release. Proper warning will be given if breaks found when automated tests are used.
  - Using TDD, should results in faster, more extensible code with fewer bugs that can be updated with minimal risks.
- **Good for teamwork**

In the absence of any team member, other team members can easily pick up and work on the code. It also aids knowledge sharing, thereby making the team more effective overall.

- **Good for Developers**

  Though developers have to spend more time in writing TDD test cases, it takes a lot less time for debugging and developing new features. You will write cleaner, less complicated code.

Use Case Diagram: A use case diagram is a graphic depiction of the interactions among the elements of a system. A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. Use case diagrams are employed in UML (Unified Modeling Language), a standard notation for the modeling of real-world objects and systems.



Use Case diagram to Unit Test: Here, this is a leave management use case diagram. We will show how to split requirements from a diagram and turned into unit test.

TDD (Test-Driven Development)

Apply For Leave:

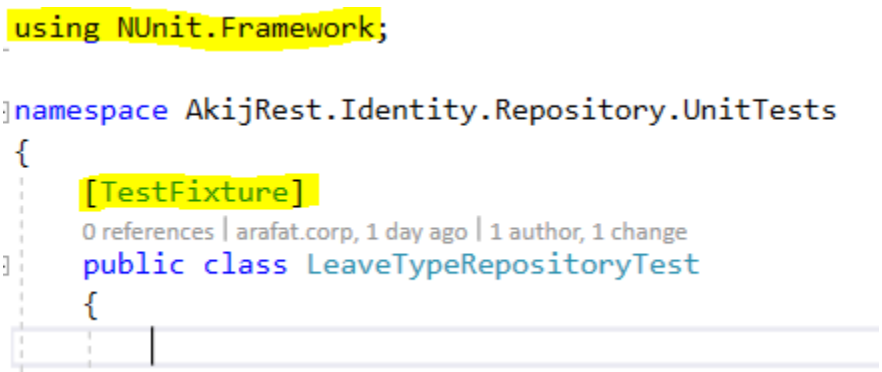
- Get permitted Leave Type
  - Check Permission
    - Create
    - Update
    - Get
    - Delete(if Possible)
  - Get Leave Type
    - Create
    - Update
    - Get
    - Delete(If Possible)
- Get Available Leave for Every Type
- Check policy Configuration for at this moment availability
  - Create
  - Update
  - Get
  - Delete(If Possible)
- Apply for leave
  - String To Date Time Convert
  - Save
  - Modify
  - Get
  - Delete(Not Applicable)

Now this is the breakdown of Apply for leave process. Here every root components will be the Test Method and the last parent of root will be test class. So in this situation Permission, Leave type, Leave Policy, and Leave will be the test class and there child crud operation will be the Test Method.

Unit Tests in our ADA projects:

TestFixture:

We have used NUnit and NUnit3TestAdapter nuget package for our unit tests.



TDD (Test-Driven Development)

We have used Nunit.famework;

Here [TestFixture] is use to determine a test class. This testFixture attribute will represent a class as a Test class.
We write a test class for testing our production class methods as same as production class with post fix named test. Here LeaveTypeRepositoryTest class is actually for testing our LeaveTypeRepository class.

Setup:

Here, LeaveTypeRepository is a class which object is initialize on a method named Setup. We use an annotation [SetUp] that represents a setup method which is called before any test method.

```
private LeaveTypeRepository _obj;
[SetUp]
0 references | arafat.corp, 1 day ago | 1 author, 1 change
public void Setup()
{
    _obj = new LeaveTypeRepository();
}
```

This is one kind of constructor for test class.

TearDown: Here, TearDownTest method kind of destructor method. Annotation [TearDown] is used to call this method at the end of the test in this test class. Generally it used to clear all object which is used in the test.

```
[TearDown]
0 references | arafat.corp, 1 day ago | 1 aut
public void TearDownTest()
{
    _obj = null;
}
```

TDD (Test-Driven Development)

Test: Tests methods are the methods where we have to write test code to test another production method. Test method is determined by the annotation [Test].To write a test method we follow a

```
[Test]
0 references | arafat.corp, 2 days ago | 1 author, 2 changes
public void ToDateTime_StringToDateTime_ReturnDateTime()
{
    //Arrange
    string date = "31/12/2018"; //default format in dd/MM/yyyy
    //Act
    var result = Datetime.ToDateTime(date);
    //Assert
    Assert.AreEqual(new DateTime(2018,12,31), result);
}
```

standard rules. Test Method name syntax is ProductionethodName_scenario of this method_ Retuns.

Here, ToDateTime is the production method name. Below production method.

```
public static DateTime ToDateTime(string date)
{
    DateTime dateTime = DateTime.ParseExact
    (
        date,
        GetStringFromDateFormat(Format.dd_MM_yyyy),
        System.Globalization.CultureInfo.InvariantCulture
    );
    return dateTime;
}
```

Scenario part should be the main task of production method and return part should be the return type of production method. Here, scenario is StringToDateTime and return is DateTime.

There are three part of a method. Arrange, Act and Assert are the three part of a test method.

- Arrange part is to initialize and ready data for test
- Act execute the production method with data
- Assert take a decision whether is it pass or fail with return result.

This is also called the AAA pattern for a test method.

TDD (Test-Driven Development)

TestCase: If we want to test a method by passing multiple input then we can use this [TestCase()] annotation.

```
[TestCase(1)]
[TestCase(2)]
[TestCase(3)]
0 references | arafat.corp, 1 day ago | 1 author, 1 change
public void Get_GetLeaveTypeById_ReturnListOfLeaveType(int id)
{
```

Summary:

- Test-driven development is a process of modifying the code in order to pass a test designed previously.
- It more emphasis on production code rather than test case design.
- It is sometimes known as **"Test First Development."**
- TDD includes refactoring a code i.e. changing/adding some amount of code to the existing code without affecting the behavior of the code.
- TDD when used, the code becomes clearer and simple to understand.