**STUB vs FAKE vs SPY vs MOCK**

**Mocks:**

* Mocking is basically a method of surpassing any bunch of function in a class for unit testing.
* The behaviour of the mocked interface can be changed dynamically based on scenarios.
* Mock gives the full control over the behaviour of the mocked object.
* We should first import the following line for mocking:

Using Moq

* We define a mock object using the following code:

Mock<checkEmployee> chk = new Mock<checkEmployee>();

* Maq has a Setup() function by which we can set up the mock object.

chk.Setup( X => X.checkEmp() ).Return(true);



LAMBDA Expression



* Whenever the unit test application encountered the checkEmp() function will always return true without executing its code.

**Fake Framework:-**

* It helps us isolating the code we have testing by replacing other parts of the application with *stubs* and *shims*.
* The *stubs* and *shims* are pieces of codes that are under the control of our tests.

**Stubs:-**

* + It replaces a class with a small subtitle that implement the same interface. It returns the predefined output regardless of the input.
  + Stubs cannot used for static method or sealed type.
  + Stops provide implementation of interface and abstract method that can be used in testing.

**Shims:-**

* + It modifies the compiled code of our application at runtime show that instead of making a special method call it returns the shim code that our test provides.
  + Shim Can be used to replace calls to assembly that you cannot modify such as.net assembly and system.dll
  + SHIM is slower as it rewrites code at runtime.
  + SHIM can replace calls to private methods if all the type on the method signature are visible.
* Below is a code explaining how we use fake for testing while mocking of code:

public int GetContosoPrice(IStockFeed feed) => feed.GetSharePrice("COOO");

[TestClass]

class TestStockAnalyzer

{

[TestMethod]

public void TestContosoStockPrice()

{

**// Arrange:**

// Create the fake stockFeed:

IStockFeed stockFeed =

new StockAnalysis.Fakes.StubIStockFeed() // Generated by Fakes.

{

// Define each method:

// Name is original name + parameter types:

GetSharePriceString = (company) => { return 1234; }

};

// In the completed application, stockFeed would be a real one:

var componentUnderTest = new StockAnalyzer(stockFeed);

**// Act:**

int actualValue = componentUnderTest.GetContosoPrice();

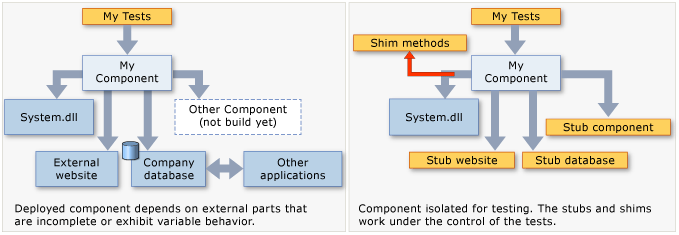
**// Assert:**

Assert.AreEqual(1234, actualValue);

}

...

}



**SPIES:-**

* It is advanced version of fake which can store the previous state of the object
* It can be useful to mimic the retry services or to check the scenarios like text “if the function called at least once”.
* We can also create a spy for logger to store and validate all the log logged while running the test case.

**PRIORITY:-**

**STUB -> FAKE -> SPY -> MOCK**

* Stub is the lightest in the most static version of this chain.
* Fake is more powerful than stub.
* Spy is an advanced version of the fake which can store the previous state of the object.
* mock is the most powerful and flexible version in the chain.

**Which one to use where:-**

* Use Fake and Spies to mimic the behaviour of business interface or services like retry, logging etc.
* Use Fakes when you want a re-usable concrete implementation that works similar to the real implementation with re-usability across tests (e.g. In Memory Database).
* Use Stubs to represent database objects.
* Use Stubs when you want a hardcoded response/implementation that will be re-used across tests.
* Use Mocks when you need dynamic responses for individual tests, that may not necessarily require re-usability across tests.
* Mocks sometime make the test case difficult to read and difficult to understand.
* Improper use of mock may impact test strategy in a negative way.