MOQ Documentation

Paul Wroe, Nate Eckardt, and Brent Stewart

April 2nd, 2013

Introduction:

Unit testing specific classes can become difficult or impossible to test when relying on additional external classes. Unit testing with external classes can have inconsistent results, or be difficult to instantiate. For example, a random number generator class will provide unpredictable data to the test. Moq addresses many of these issues by creating a mock instance of the object.

Moq is a plugin for Visual Studios. It is a tool that allows some objects to pretend to be an instance of an external dependency. This allows otherwise untestable functions to be tested in C#.

Moq is used to mock interfaces not classes. Any class's function that are going to be mocked must have a function that is defined in an implemented interface.

There are known limitations to Moq. With Moq, returns on static methods cannot be mocked, neither can classes that do not implement an interface. There are some workarounds for these issues. Classes can be wrapped and then implement an interface of the wrapper, or functions can be extracted and set to virtual, allowing Moq to do a partial mock. The later is better illustrated in the example of Moq untestable code.

Using Moq:

1) Identify all of the external dependies contained in the class.
    a. Objects not a part of the class need to be either instantiated or mocked. Otherwise, the class being tested will throw null pointer exceptions.
    b. Ensure the objects that need to be mocked implement an interface. Make sure the function being called is indeed implemented from said interface.
        i. If there is no interface implemented, make one if possible.
        ii. If there is no way to implement an interface, create a wrapper class
            1. Make sure to have your wrapper class implements an interface.
        iii. When there is no way to create a wrapper class, consider alternatives.
            1. Create a virtual function for the line that cannot be tested.
            2. Consider not testing the function.
            3. Use a supplemental testing suite.
        iv. If this is at all confusing, good! There are examples further down.

    c. Check to make sure you can use setter injection for the mock object.
2) Create a mock using Moq.
    a. Mock<InterfaceToMock> mock = new Mock<InterfaceToMock>();
    b. classToTest.interface = mock.Object;
        i. mock.Object exposes the mock Object instance.
3) Call methods on mocked object, one of many is Setup >> Returns
    a. mock.Setup(m => m.FunctionToMock()).Returns(objectOfExpectedType);
        i. lambda expressions …
            1. can write local functions that can be passed as arguments or returned as the value of function calls
            2. If you want to learn more on why a lambda is used…
                a. http://msdn.microsoft.com/en-us/library/bb397687.aspx
4) Create mock for virtual function (side step code you cannot test)
    a. Extract untestable code into virtual function.
    b. Mock<ClassToTest> mock = new Mock<ClassToTest>();
        i. Using the same Setup >> Returns as from item 3 above.
            1. Mock.Setup(m => m.VirtualFunctionToMock()).Returns(objectOfExpectedType);
    c. To have the code run the test correctly
        i. Use the mock as the execution point
            1. mock.Object.MethodToTest();
                a. if confusing, look at the virtual method example

EXAMPLES

Class for a normal mock:

Code block 1

```
public class ManageUserController : Controller, IManageUserController
{
        public IManageUserService manageUserService { get; set; }

        public ActionResult Delete(int id = 0)
        {
            bool isSuccess;
            User user = manageUserService.GetUser(id, out isSuccess);
            if (user == null)
            {
                return HttpNotFound();
            }
            return View(user);
        }

       public ManageUserController()
       {
              manageUserService = new ManageUserService();
       }
}

public interface IManageUserService
{
       // the mock will overrride this method call!
       // allowing the user to return a wanted value
        User GetUser(int id, out bool isSuccess);
}
```

EXAMPLE ONE OF MOQ USING CODE BLOCK 1:

This test does not use a lambda expression.

```
[TestMethod]
public void ManageUserDetailsFailTest()
{
	bool isSuccess;

	// instantiate instance of class
	ManageUserController manageUserController = new ManageUserController();

	// create mock of IMangeUserService
	Mock<IManageUserService> mockManageUserService = new Mock<IManageUserService>();

	// Set mock to object in class
	manageUserController.manageUserService = mockManageUserService.Object;

	// Set wanted return  on the mock
	mockManageUserService.Setup(m => m.GetUser(0, out isSuccess)).Returns((User)null);

	// No object made, so will be a null user, therefore execute return
	// HttpNotFound();
	Assert.IsInstanceOfType(manageUserController.Details(USER_FOUR),
	typeof(HttpNotFoundResult));

}
```

EXAMPLE TWO OF MOQ USING CODE BLOCK 1:

This test does use use a lambda expression.

```
 [TestMethod]
public void ManageUserDetailsPassTest()
{
	bool isSuccess;
	User userTest = new User();
	ManageUserController manageUserController = = new ManageUserController();

	// create mock of IMangeUserService
	Mock<IManageUserService> mockManageUserService = new Mock<IManageUserService>();

	// Set mock to object in class
	manageUserController.manageUserService = mockManageUserService.Object;

	// Set wanted return on the mock
	// here is the lamdba.
	mockManageUserService.Setup(mockObject => mockObject.GetUser(USER_FOUR, out
	isSuccess)).Returns(userTest);

	// Since returns object of not null, executes return View(user);
	Assert.IsInstanceOfType(manageUserController.Details(USER_FOUR),
	typeof(ViewResult));
}
```

Moq with Untestable code!

CODE BLOCK 2:

```
public class ManageUserService : IManageUserService
{
        public IPracticeGDVPDao context { get; set; }
        public List<User> GetAllUsers(out bool isSuccess)
        {
            isSuccess = false;
            // the next line of code had read…
            // context.Users().Include(u => u.webpages_Roles).ToList();
            // However, ToList() is a static method, which cannot be mocked
            // so instead, the line was extracted into a new method,
            // GetAllUsersList();
            List<User> usersList = GetAllUsersList();
            if (usersList.Count > 0)
            {
                isSuccess = true;
            }
            return usersList;
        }

       // The method call has been moved to a virtual method,
       // which can be partially mocked using Moq!
        public virtual List<User> GetAllUsersList()
        {

              // IN THE TEST, context does not need to be instantiated
              // as this function will be mocked.
            return context.Users().Include(u => u.webpages_Roles).ToList();
        }
}
public interface IPracticeGDVPDao : IDisposable
{
       // the mock will overrride this method call!
       // allowing the user to return a wanted value
       IDbSet<User> Users();
}
```

EXAMPLE 4 using Code Block 2:

```
public void GetAllUsersPassTest()
{
	User user = new User();
	List<User> listOfUsersTest = new List<User>();
	listOfUsersTest.Add(user);
	bool isSuccess;

	// set mock of ManageUserService which allows to return object type GetAllUsers()
	Mock<ManageUserService>() mockManageUserService = new Mock<ManageUserService>();

	// set object to return
	mockManageUserService.Setup(m => m.GetAllUsersList()).Returns(listOfUsersTest);

	// assert object is of type exptec and success!
	Assert.AreSame(listOfUsersTest, mockManageUserService.Object.GetAllUsers(out
	isSuccess));
	Assert.IsTrue(isSuccess);
}
```

EXAMPLE 5 using Code Block 2:

```
public void GetAllUsersFailTest()
{
	bool isSuccess;
	List<User> listOfUsersTest = new List<User>();

	// set mock of ManageUserService which allows to return object type GetAllUsers()
	Mock<ManageUserService>() mockManageUserService = new Mock<ManageUserService>();

	// needed to test Lambda expression
	mockManageUserService.Setup(m => m.GetAllUsersList()).Returns(listOfUsersTest);

	// verify object is of expected type, and fail (list is empty!)
	Assert.AreSame(listOfUsersTest, mockManageUserService.Object.GetAllUsers(out
	isSuccess));
	Assert.IsFalse(isSuccess);
}
```