

The role of Mocking in TDD, Test First with Rhino Mocks, KandaAlpha ^[1]

Recently I got into a healthy discussion ^[2] with Keith Patton ^[3] about Constructor Injection and other such topics to do with Test Driven and Domain Driven Design.

One way or another this lead to the birth of Kanda Alpha ^[4] with the goal of demonstrating best practice domain driven design utilising Visual Studio 2010 Beta, ASP.NET MVC 1.0 and Db4o ^[5] as the data store.

I've decided to give an explanation as to the benefits of Mocking and the Test First approach.

Mocking quite simply allows you to test components in isolation from their dependencies. By mocking out the results and behaviour of the dependent components you can focus on verifying the behaviour of a single class.

Where I've found it most useful is in designing the service layer and this is usually the first thing I design after hashing out the Domain Entities.

There has also been some comments on Keith's blog about the purpose of the Service layer if all it is doing is acting as a Facade to the Repository. This is a valid point and I'm going to try and give a real life example which demonstrates that this isn't the case.

From Martin Fowler's Domain Driven Design

"A Service Layer defines an application's boundary [Cockburn PloP] and its set of available operations from the perspective of interfacing client layers. It encapsulates the application's business logic, controlling transactions and coordinating responses in the implementation of its operations."

<http://martinfowler.com/eaCatalog/serviceLayer.html> ^[6]

Second only to the Entities Model the Service Layer is the heart of your application so should be the focus of the design and that's where Mocking will add value.

The example I'm going to demonstrate is the very simple but very common Registration of a User.

When a User is created there are a number of requirements that need to be met:

- Geocode the Users Address

- Persist the User to a data store
- Send an Email asking them to confirm their email

Right so let's see some code.

Firstly I will define the IRegistration interface

[7]

```
public interface IRegistrationService
{
    /// <summary>
    /// Validate the supplied User object,
    /// geocode the address if it is supplied
    /// then persist to datastore and
    /// finally send confirmation email
    /// </summary>
    /// <param name="user"></param>
    /// <returns></returns>
    bool RegisterUser(User user);
}
```

[8]

Ok as we are doing Test first let's create an Implementation and set up our first Unit Test.

[9]

```
public class RegistrationService : IRegistrationService
{
    #region IRegistrationService Members

    public bool RegisterUser(User user)
    {
        throw new NotImplementedException();
    }

    #endregion
}
```

[10]

Now the Mocking framework of choice is Rhino.Mocks ^[11].

So here is the very first Unit Test.

[12]

```
[TestMethod]
public void RegisterUser_Pass()
{
    var mocks = new MockRepository();

    var registrationService = new RegistrationService();

    var user = new User
    {
        FirstName = "Will",
        LastName = "Beattie",
        Email = "beattie.w@gmail.com"
    };

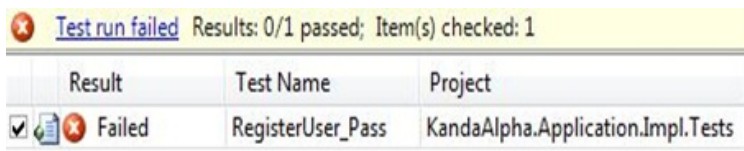
    var actual = registrationService.RegisterUser(user);

    Assert.IsTrue(actual);
}
```

[13]

Note this test will fail as RegisterUser has not been implemented yet, this is fine as it is failing for a very valid reason.

[14]



[15]

Because of the requirements listed above the IRegistrationService implementation is going to have dependencies on some other components to fulfil these requirements.

- IUserRepository
- IAddressGeocoder
- IMessageSender

So let's define those

[16]

```
public interface IAddressGeocoder
{
    /// <summary>
    /// Takes the supplied address and returns the coordinates
    /// </summary>
    /// <param name="address"></param>
    /// <returns></returns>
    Coordinates GetCoordinatesFrom(string address);
}
```

[17]

Now we will define the IUserRepository which will be used to persist the User.

[18]

```
public interface IUserRepository
{
    /// <summary>
    /// Persist the supplied User
    /// to the datastore
    /// </summary>
    /// <param name="user"></param>
    void Add(User user);
}
```

[19]

Finally we need a component to send the confirmation email.

[20]

```
public interface IMessageSender
{
    /// <summary>
    /// Use the supplied User object to construct
    /// the registration email then send.
    /// </summary>
    /// <param name="user"></param>
    /// <returns></returns>
    bool SendConfirmationMessage(User user);
}
```

[21]

Ok now that we have defined the Interfaces for our dependencies, let's add them to our RegistrationService implementation. For this I am going to use Constructor Injection ([link here](#)) it will become clear shortly why.

[22]

```
public class RegistrationService : IRegistrationService
{
    private readonly IAddressGeocoder addressGeocoder;
    private readonly IUserRepository userRepository;
    private readonly IMessageSender messageSender;

    public RegistrationService(IAddressGeocoder addressGeocoder,
        IUserRepository userRepository,
        IMessageSender messageSender)
    {
        this.addressGeocoder = addressGeocoder;
        this.userRepository = userRepository;
        this.messageSender = messageSender;
    }

    #region IRegistrationService Members

    public bool RegisterUser(User user)
    {
        throw new NotImplementedException();
    }

    #endregion
}
```

[23]

Now that we have obviously broken our test because there is no longer a default constructor. So let's fix up our test so that it runs again.

[24]

```
[TestMethod]
public void RegisterUser_Pass()
{
    var mocks = new MockRepository();
    var messageSender = mocks.Stub<IMessageSender>();
    var userRepository = mocks.Stub<IUserRepository>();
    var addressGeocoder = mocks.Stub<IAddressGeocoder>();

    var registrationService =
        new RegistrationService(addressGeocoder,
                                userRepository,
                                messageSender);

    var user = new User
    {
        FirstName = "Will",
        LastName = "Beattie",
        Email = "beattie.w@gmail.com"
    };

    var actual = registrationService.RegisterUser(user);

    Assert.IsTrue(actual);
}
```

[25]

You will notice above that I have used the MockRepository to generate in-memory Stubs. We do this because we don't want to worry about the implementations at this stage.

Now we need to Setup some results.

[26]

```
[TestMethod]
public void RegisterUser_Pass()
{
    var mocks = new MockRepository();
    var messageSender = mocks.Stub<IMessageSender>();
    var userRepository = mocks.Stub<IUserRepository>();
    var addressGeocoder = mocks.Stub<IAddressGeocoder>();

    var registrationService =
        new RegistrationService(addressGeocoder,
            userRepository,
            messageSender);

    var address = "1 Queen Street, Auckland City, Auckland, New Zealand";
    var user = new User
    {
        FirstName = "Will",
        LastName = "Beattie",
        Email = "beattie.w@gmail.com",
        Address = address
    };

    //let's set up the results
    using (mocks.Record())
    {
        SetupResult
            .For(addressGeocoder.GetCoordinatesFrom(address))
            .Return(new Coordinates { Latitude = 30.000, Longitude = 30.000 });

        SetupResult
            .For(messageSender.SendConfirmationMessage(user))
            .Return(true);
    }

    var actual = registrationService.RegisterUser(user);

    Assert.IsTrue(actual);

    userRepository.AssertWasCalled(ur => ur.Add(user));
    addressGeocoder.AssertWasCalled(ag => ag.GetCoordinatesFrom(address));
    messageSender.AssertWasCalled(ms => ms.SendConfirmationMessage(user));
}
```

[27]

Now that we have setup our test we can write the implementation, which should be easy.

[28]

```
public bool RegisterUser(User user)
{
    if (user == null)
    {
        throw new ArgumentNullException("user");
    }

    var coordinates = addressGeocoder.GetCoordinatesFrom(user.Address);

    user.Coordinates = coordinates;

    userRepository.Add(user);


    var sent = messageSender.SendConfirmationMessage(user);

    return sent;
}
```

[29]

Now running the test should give us the all important green tick.

[30]

Result	Test Name	Project
 Passed	RegisterUser_Pass	KandaAlpha.Application.Impl.Tests

[31]

Well that's it. By following this approach you are forced to think about how classes interact with other classes and as a result you end up with loosely coupled components that are Unit Testable.

The source code is available to on the KandaAlpha ^[32] project on Codeplex ^[33].

1. <http://blog.willbeattie.net/2009/07/role-of-mocking-in-tdd-test-first-with.html>
2. <http://blog.keithpatton.com/2009/06/25/db4o+POCO+Repository+Using+Visual+Studio+2010+Net+40+Beta+1+And+ASPNet+MVC+10.aspx>
3. <http://blog.keithpatton.com/>
4. <http://kandaalpha.codeplex.com/>
5. <http://www.db4o.com/>
6. <http://martinfowler.com/eaCatalog/serviceLayer.html>
7. http://lh4.ggpht.com/_la3Rv-0jOCI/Skz7xEsfDI/AAAAAAAAAS4/OT_fjPlubig/s1600-h/regservice%5B9%5D.jpg
8. http://lh4.ggpht.com/_la3Rv-0jOCI/Skz7xEsfDI/AAAAAAAAAS4/OT_fjPlubig/s1600-h/regservice%5B9%5D.jpg
9. http://lh6.ggpht.com/_la3Rv-0jOCI/Skz7y1_a8OI/AAAAAAAAATA/ImdcwlsNsg/s1600-h/regserviceimpl%5B4%5D.jpg
10. http://lh6.ggpht.com/_la3Rv-0jOCI/Skz7y1_a8OI/AAAAAAAAATA/ImdcwlsNsg/s1600-h/regserviceimpl%5B4%5D.jpg
11. <http://ayende.com/projects/rhino-mocks.aspx>
12. http://lh4.ggpht.com/_la3Rv-0jOCI/Skz70RxGWXI/AAAAAAAAATI/bw6zabAqH-k/s1600-h/regusertest%5B4%5D.jpg
13. http://lh4.ggpht.com/_la3Rv-0jOCI/Skz70RxGWXI/AAAAAAAAATI/bw6zabAqH-k/s1600-h/regusertest%5B4%5D.jpg
14. http://lh4.ggpht.com/_la3Rv-0jOCI/Skz71t-9PnI/AAAAAAAAATQ/8H2yuAtweZk/s1600-h/reguserfail%5B3%5D.jpg
15. http://lh4.ggpht.com/_la3Rv-0jOCI/Skz71t-9PnI/AAAAAAAAATQ/8H2yuAtweZk/s1600-h/reguserfail%5B3%5D.jpg
16. http://lh6.ggpht.com/_la3Rv-0jOCI/Skz73Hwlh_I/AAAAAAAAATY/8DBctDQ823U/s1600-h/geoservice%5B6%5D.jpg
17. http://lh6.ggpht.com/_la3Rv-0jOCI/Skz73Hwlh_I/AAAAAAAAATY/8DBctDQ823U/s1600-h/geoservice%5B6%5D.jpg
18. http://lh3.ggpht.com/_la3Rv-0jOCI/Skz74tnwFI/AAAAAAAAATg/nhMB8qC_Uhk/s1600-h/userRep%5B5%5D.jpg
19. http://lh3.ggpht.com/_la3Rv-0jOCI/Skz74tnwFI/AAAAAAAAATg/nhMB8qC_Uhk/s1600-h/userRep%5B5%5D.jpg

20. http://lh5.ggpht.com/_la3Rv-0jOCI/Skz76D_o0WI/AAAAAAAAATo/Yv-P3Z2XqZI/s1600-h/messageSender%5B4%5D.jpg
21. http://lh5.ggpht.com/_la3Rv-0jOCI/Skz76D_o0WI/AAAAAAAAATo/Yv-P3Z2XqZI/s1600-h/messageSender%5B4%5D.jpg
22. http://lh4.ggpht.com/_la3Rv-0jOCI/Skz77n_tgFI/AAAAAAAAATw/XH4ECMy5wJw/s1600-h/regwithdependencies%5B3%5D.jpg
23. http://lh4.ggpht.com/_la3Rv-0jOCI/Skz77n_tgFI/AAAAAAAAATw/XH4ECMy5wJw/s1600-h/regwithdependencies%5B3%5D.jpg
24. http://lh6.ggpht.com/_la3Rv-0jOCI/Skz79Y3c9KI/AAAAAAAAAT4/kbmJPbrkeoI/s1600-h/regservicetestwithmocks%5B7%5D.jpg
25. http://lh6.ggpht.com/_la3Rv-0jOCI/Skz79Y3c9KI/AAAAAAAAAT4/kbmJPbrkeoI/s1600-h/regservicetestwithmocks%5B7%5D.jpg
26. http://lh4.ggpht.com/_la3Rv-0jOCI/SIL4o1oG5_I/AAAAAAAAAAUI/-wljcMSwFe0/s1600-h/regserviceStubs%5B4%5D.jpg
27. http://lh4.ggpht.com/_la3Rv-0jOCI/SIL4o1oG5_I/AAAAAAAAAAUI/-wljcMSwFe0/s1600-h/regserviceStubs%5B4%5D.jpg
28. http://lh5.ggpht.com/_la3Rv-0jOCI/SIL4qTuBm4I/AAAAAAAAAUQ/JiSgpE7m4i0/s1600-h/reguserimpl%5B3%5D.jpg
29. http://lh5.ggpht.com/_la3Rv-0jOCI/SIL4qTuBm4I/AAAAAAAAAUQ/JiSgpE7m4i0/s1600-h/reguserimpl%5B3%5D.jpg
30. http://lh6.ggpht.com/_la3Rv-0jOCI/SIL4sva5h-I/AAAAAAAAAUY/KggvwfX-GTc/s1600-h/reguserpass%5B4%5D.jpg
31. http://lh6.ggpht.com/_la3Rv-0jOCI/SIL4sva5h-I/AAAAAAAAAUY/KggvwfX-GTc/s1600-h/reguserpass%5B4%5D.jpg
32. <http://kandaalpha.codeplex.com/>
33. <http://kandaalpha.codeplex.com/>