Let’s start off with, what is unit testing?

Paraphrasing from a Wikipedia article: Unit testing is a software testing method by which individual units of source code...are tested to determine whether they are fit for use.

I feel one of the key lines in this is individual units of source code. This is something I’ve found time and again to be hard to define. Here is the way that I try to conceptualize it. As we are building software, we create seams, areas where we put together more than one thing to build our software. These seams are what I try to focus on when I test. A seam implies that you have boundaries, and this is where those boundaries come together. For instance, looking at the architecture of an application, there are three main areas of development: Data Store, Business Logic and User Interface. Each is bound to its area of application, but all need to come together to create the whole application.

For me, Unit Testing are tests that are created to test something within one of those bounds and Integration Testing is where tests are created to cross those bounds.

Now, to dig into some code. As a starting point, lets assume we need to write our version of c#’s IsNullOrWhiteSpace function. If we were following a TDD approach, we would write one or many tests initially that we feel would adequately demonstrate that our implementation is working as expected. A lot of times this is done with a specific design in mind and in many instances, working through it from this perspective can influence the design of the development as well. With c#, having a skeleton design initially is extremely beneficial in writing the tests as you will be able to utilize Intellisense.

In testing, you will hear of the three A’s, which stands for Arrange, Act, Assert. Arrange – set up the test data and any mock implementations. Act – perform the action under test and receive the result. Assert – validate that the result is the expected one.

Go through the test cases…  
see if they make sense…use random data or not?  
any missing?

Implementation suggestions?  
Iterate…

That’s the basics of unit testing, any questions?

Next step would be a bit more realistic. Going back to our application, lets now look at the business logic layer. Lets look at a generic ‘save’ method. Lets assume there are things that would make the item to save either valid or invalid and we only want to save it if it is valid. The business logic class would have a data store, take in an object, validate it and then act appropriately based on that validation. We would like to test that this logic works, but we want our tests to run in environments where we don’t want to have to setup SQL to work with it and/or we don’t want to pollute our data store with fake data. This is where one of the powerful gains of programming to an interface can come in to play(monkey patching/duck punching/duck typing can do similar things in dynamic languages like javascript). If we place the data store behind an interface, we can test the business logic without needing an actual data store.

We create a mock data store and as part of our setup, tell it which methods we expect to be called, what parameter value(s) we expect it to be called with and what it should return when called with those parameters. You can be generic or specific with these expectations. We can walk through a few iterations.

Can anyone spot something we aren’t testing? (ConvertToContract -> a.k.a. the return object from the save method)