So there are three ways to use dependency injection. We can inject dependencies via method parameters, by properties, and by constructors. In this lecture, I'm going to show you how to inject dependencies via method parameters. So here in our read video title method, instead of working with this concrete implementation FileReader, I'm going to pass an IFileReader object as a parameter to this method. Call this FileReader and then we can get rid of this new operator.

So we use this fileReader argument. With this simple change, our video service class becomes loosely coupled and testable. Because in our production code, we can pass a real fileReader object to this project, whereas in our test, we can pass a fake fileReader object. Let me show you. So, back in our solution explorer, in the mocking folder, I'm going to temporarily add a class called Program. Here I'm going to add a public

static void Main method, so this video service we have here, most likely we're going to this somewhere in our application, and I want to simulate that in our program class. So I'm going to create a VideoService object, VideoService, and call ReadVideoTitle method.

Now here we need to pass an instance of class that implements IFileReader interface. So we can pass a real FileReader new FileReader, and with this we get a title. Now this is security for demonstration, in real world applications instead of us manually newing up this object here, we use a dependency injection framework. So a dependency injection framework is responsible for newing up these objects and passing them to our methods. We're going to look at dependency injection frameworks in just a few minutes. So this is how we pass a real object for this method, ReadVideoTitle. Now let's go ahead and write a unit test for this method, and there I'm going to show you how to pass a fake object. So back in our unit testing project,

I'm going to add a new unit test, and by the way, it's a good practice that you're unit testing project mimic the same structure you have on the project on your test. So here we have this top little folders, like fundamentals and walking, but I forgot to create these folders in this unit testing project.

So from now on, I'm going to add all the unit tests for the classes defined in the mocking name space inside the mocking filter. So, let's add a new directory with Mocking and here I'm going to add a new class called VideoServiceTests.

First I'm going to apply a test fixture attribute, and create a test public void, method on the test is called ReadVideoTitle, let's look at the implementation of this method, let's say I want to test this execution path, if video is null we expect an error message. So this fake FileReader that we implemented simply returns an empty string.

This string can not be deserialized as a video object.

Okay? So back in our test class, the scenario is EmptyFile. So, there is nothing in our file that's why we are returning an empty string in our fake FileReader object. And we expect this method to return an error message. First, let's create an instance of this service, new VideoService. Now, we're going to call service .ReadVideoTitle. But this time we're going to pass our FakeFileReader object. So new FakeFileReader. Now we get the result, and finally assert That result Does.Contain the word error. Now to make this test more reliable, I'm also going to change this IgnoreCase property,so we're going to ignore case sensitivity of strings in this

test. Let's run this. Okay, here's our test, and it successfully passed, beautiful.

So, let's quickly recap, back in our video service, in this lecture you learned how to pass or inject a dependency as a method parameter. In the next lecture, I'm going to show you how to inject the dependency using a property.