Alright, now in our unit testing project, in the Mocking folder, let's add a new test class called InstallerHelper

Tests. We decorate this with test fixture. And also create a Setup method, public void method, SetUp, and decorate this with the SetUp attribute. Now here we need to mock IFileDownloader interface. So, I'm going to create a private field called \_file Downloader and initialize it to a new instance of Mock of IFileDownloader.

Now here we can press alt and enter to create a private field by this name. So, here's our private field, of type mock of IFileDownloader, okay? Now similarly, we need an installer helper. So, Installer Helper is a new InstallerHelper that takes a FileDownloader object so we pass fileDownloader.object.

And then, use the IDE to create this private field. Beautiful. So, our setup is ready now let's create our first test method. So Test public void, the method is called DownloadInstaller.

The first scenario is where download fails, so, DownloadFails, this method should Return False. Now in the arrange part we need to program our mock. So, fileDownloader, we use the Setup method, now we pass a lambda expression, fileDownloader goes to when we call the DownloadFile method, and give it two arguments, in this case I'm going to pass empty string and I will tell you why in a second. Then this method, should throw an exception. So throws a web exception because that's the type of exemption we have in our catch block here, okay.

So here's the arrange part, now we need to act, so installerHelper. DownloadInstaller. Here we need to pass two arguments. The first one is customer and the second one is installer name. So I can pass a and b as simple values, but someone looking at this code, they don't know exactly what these values are. So in this case I'm going to break the rules that I told you about earlier, and use a descriptive name, so customer, and installer. This makes our test more real (?)

Now we get the result, and Assert that result Is False. Let's run the test and see what happens. Our test failed. We expected false, but we got true. How did that happen? Well, something interesting about these mock objects you need to know, is that when you program them using the set up method, this behavior you define in this case, throwing a web exception only happens when we call this download file method to exact same arguments. In other words, if you pass any other argument, this behavior will not happen.

So that method will not do anything. In this case, in our installer helper class, this is the argument that we are passing for a FileDownloader. So, to fix our test, we need to replace this empty string with a url like this, httpexample.com/customer/installer.

And for the second argument, instead of an empty string, we should use null. Because if you look at installer helper, the second argument is setupDestination File, and this field is not initialized anywhere. So by default, it's null. Now, back to our test. With these arguments, if you run this test, it should pass. Let's verify that.

Okay, beautiful, it passed. However, I don't quite like the way we have written this test. It's getting a little bit noisy, it's long (?) string. And this is a simplified example. Sometimes in your applications you may not even have access to that argument that is sent to a given method like DownloadFile. In that case, you want to program this mock in more generic way. So, instead of this first argument, we can pass It, this is one of the classes defined in mock. It has a method called IsAny which is generic, then we pass a string. Similarly, I'm going to use this for the second argument, so it is any string. Like this.

Now, our mock is more generic. So when we call DownloadFile, with any arguments, it's going to throw this web exception, okay? Now finally, let's reformat this code to make it a little more readable. Like this. Let's run the test one more time.

It's passing. Beautiful. Now, the second test. So, to save time, I'm going to duplicate this method, change the scenario from download fails to download completes, and we expect to get true. So here we don't need to program our mock, because by default when we call this download file method, it doesn't do anything, it doesn't throw an exception, and we don't expect a value here either. So, let's delete that.

And finally let's Assert that result is true. Let's run the test.

Beautiful. Now here's a question for you. You might ask, Mosh, here we are writing a test for this web exception and we Assert that result should be false. What if this download file method throws a different kind of exception at run time. Should we write another test and program our mock to throw a different kind of exception? No, we don't have to do that. Because the whole purpose of this method. is to return true if the download completes, and false, if download fails. So web exception is specifically the kind of exception that may happen when downloading a file. If you get a different kind of exception, let's say null reference exception, or argument out of range exception, or divide by zero exception, we don't want to hide that exception by returning false.

So we want that exception to be propagated, perhaps there is an exception logger somewhere programmed globally, so that's going to get that exception and log it somewhere. Only want to return false if we get a web exception, and for that very reason we don't need to write another test for this method.