So, because this code touches an external resource, we need to do some refactoring. And open up this class for testability. So, back in our solution in the Mocking folder, I'm going to create a new class, called FileDownloader.

You want to isolate all the code, but downloading the file inside this class. So here we need a simple method, public\_void, DownloadFile that takes two parameters. One is the url of the file, and the other is the path and the file system to store it.

Now, back in installer helper, so these are the lines that we need to move into that new class. But I'm not going to cut this code from here yet. I'm simply going to copy it, and you will see why in a second. So copy, paste it here. Alright first we need our web client object.

So, back to our helper class, I'm going to cut this first line and move the the WebClient object into our new class. Let's import NameSpace. Beautiful. So, here we need to make a couple simple changes. I'm not going to use this customer name and installerName here.

Because these concepts do not fall within the scope of FileDownloader

they belong installer helper. Because this class knows bout customerNames, it knows about installerNames this is a higher level class in our application. FileDownloader in contrast is a low level class, it's dumb,

it knows nothing about our business domain, all it knows about is how to get a file from this url and store it at this location

Okay? So, instead of using string format to construct the URL. I'm going to pass the url argument, that we get here, and also as a second argument, I'm going to use this path (?). Now let's put them on one line, so basically all we're doing is here is encapsulating the code that touches an external resource inside it's FileDownloader.

And the reason we're doing this is to extract an interface so we can mock this interface in our unit tests. I'll tell you something. If this web client class was designed properly, we didn't have to create a separate class like FileDownloader in order to extract an interface.

Sometimes when you work with third party libraries, you may notice that some of their classes come with interfaces. For example, if we did have an interface like IWebClient, then we can simply inject that interface

inside installerHelper, and then we didn't have to create an extra class like this.

So, let's remove this, now, let's extract an interface here, refactor, extract interface, and let's select this method, next, beautiful. So we have IFileDownloader. Next, we're going to inject this into the constructor, of installerHelper.

So, constructor, Inject File (?)IFileDownloader, call it fileDownloader.

And next, we're going to create and initialize the private field. So, look we have this private read on the field of type IFileDownloader, and this is initialized using the argument that we get from the constructor, basic dependency injection.

Now finally, in DownloadInstaller method, instead of using this client object, we're going to use fileDownloader, and the rest of the code is fine.

So this method DownloadInstaller is a higher level method, is delegating the task of downloading the file for this fileDownloader object, but it knows about customer names and installerNames, so it uses this concept to construct a URL. If everything goes well, it's going to return true, if something goes wrong it's going to return false. So here we have two execution paths, and we need to write two unit tests for this class.

And, that's what I'm going to show you next.