Alright, now let's see how we can write unit tests for this method. So, as we can see over these few lines, we are going in the database and getting the videos that are not processed. So this is where we're touching an external resource. So as you learned earlier, we need to extract these few lines, put them in a separate class and isolate that class from the rest of the code here.

So, back in our project, in the Mocking folder, I'm going to add a New class, so, what should we call that class? Because we are writing a query here, a pattern that we use often for encapsulating queries in our application is the repository pattern. I've explained that in my editing framework course, so we're not going to go over that again. For now, we're just going to create a new class called VideoRepository.

So this is a repository of all the videos in our application. So here we can add a new method, public IEnumberable. Why IEnumerable, because we simply want to enumerate this list. We just want to iterate over it. So back in video service, look, after we get the videos, all we need to do is to iterate over this list using this for each statement, okay? So, we simply return an IENumerable of video. Let's import a name space. GetUnprocessedVideos.

Now, back to our video service, lets select these few lines. Cut, and paste it inside our repository. Note that here we need a reference to this db context, video context. Okay? So, back in our video service, this is where we're creating that video context. Again, this is a class that touches an external resource, so, this context object should not be here. So basically, I'm going to copy this line and paste it here. So here we have, our using block, and let's put this query inside this block. Now, we need to import system.link on the top. Okay, and finally we can return videos. So you can see this video repository is encapsulating the code, that touches an external resource. Now, back to our Video Service. We no longer need this using block. Okay? So, delete. Delete. And Delete. Our code is far simpler.

So this is one of the benefits of unit testing. When you open up your classes for testability, you get better separation of concerns, and your methods end up being shorter, because you move the responsibility where it really belongs. Now here, we need to get these videos from our video repository. So the next step of refactoring is like this, that a new VideoRepository we call GetUnprocessedVideos, and get the result and store it in this variable. Okay?

However, as I told you before, when we new up the video repository here, our VideoService becomes tightly coupled to this implementation, we want to replace this implementation with an interface, with a contract, then we can use any classes that implements that interface.

So the next step is to extract an interface from our video repository.

Back here, as I showed you before, we can extract an interface from a class very easily, so here we press alt and enter, refactor, extract interface. Now, I'm going to select this method to add in the interface, next, done. So here's our new interface. Okay? Now, back to our VideoService. We need to inject that interface into this VideoService. We can inject it as a parameter to this method. We can inject it by a property or, we can inject it in the constructor here. As I explained before, construction injection is more common, so I'm going to follow the same pattern here.

So, let's add a new parameter, IVideoRepository. And we call this

repository, I'm going to follow poor man's (?) dependency injection. So, you set this to null, and initialize the video repository field here.

So first we need to create private IVideo Repository call that repository, and then set repository like this. So if we have this object here in the argument, we're going to use that otherwise we're going to new up a VideoRepository. Again, if you use a dependency injection framework, you don't have to deal with this poor man's dependency injection approach. Okay? But for this challenge it doesn't really matter. But finally, instead of newing up this VideoRepository, we're going to access the repository field. So now, our VideoService class is loosely coupled testable, next, I'm, going to show you how to write unit tests for this class.