In the unit tests we have written so far, we asserted that our methods returned the right value or set the right state. This is what we call state-based testing.

Because we test the state changes in our application. However, sometimes I'm dealing with the code that touches external resources, we need to verify the class we're testing interacts with another class

properly, this is what we call

Interaction Testing. Because we test the interaction of one class, with another. For example, here in our order service class, when we place an order this order should be written to a storage, whatever that storage is, we don't care. It can be a database or a cloud storage, one of the unit tests we need to write for the place order method is to verify that this OrderService object interacts with the storage object property.

So, we should check that our code calls the store method of the storage object with the right argument. That is the order object we receive here.

In the next lecture, I'm going go show you how to test the interaction

in between objects, however, remember to reserve this approach only for dealing with external resources. Because with interaction testing, your tests start to couple your (?) implementation. You verify that the right method is called with the right arguments. As you refactor and restructure your code, it is possible that you move some of these methods around.

And with that, you may break one or more tests. So, as I've emphasized throughout this course, your tests should test the external behavior and not the implementation. So, prefer state based testing to interaction testing. And, use interaction testing only when dealing with external resources.