A lot of people new to unit testing assume that in order to unit test a class, they should test it in isolation without any dependencies, whether those dependencies touch external resources or not. I've gone down that path and have learned the hard way, that this approach is a recipe for disaster. When you mock all the dependencies of a class, you end up with several problems. First, you see an explosion of interfaces in your application, and these interfaces serve no purpose.

For every class you have an interface, no matter what that class does. Second, your class constructors explode with too many parameters. If you have constructors like this that's often a symptom that you're mocking dependencies that should not be mocked. The third problem is that your tests start to get fat and bulky. Your set up or arranged parts get complex because they deal with so many mock objects that should be injected into the object you're testing. Plus, these tests are often very fragile, because they're tightly coupled (?) design.

As you change your implementation, these tests break frequently, because they don't test the external behavior they test the implementation. They know too much about your objects and how they interact, so here's what you need to do, use mocks as little as possible. Reserve them only for dealing with external resources. There is only one exception that you can use mocks for other cases, but for now just assume that mocks should be reserved for removing external resources from your unit test. Because otherwise your tests will get slow or fail. For example. If your application talks to a web service, you don't want that web service to be up and running when you run your unit tests.

If that web service is not available, the tests that depend on that web service will fail. And even if it's available, your tests will get slow. Unit tests should be fast. We write a lot of them. Hundreds or thousands of them, and we should run them all the time, especially as we write code and refactor our code. That's why we use mocks. To remove these external resources from our unit tests so they can run quickly and reliably. Earlier I told you about an exception for using mocks when you are not dealing with external resources. That is when you're unit testing classes with complex calculations and execution paths. Imagine class A uses class B.

Neither of these classes use any external resources, perhaps if you want to unit test A and B together, you will end up with numerous execution paths and test cases. In that case it's okay to unit test A and B independently and in isolation. When unit testing A, you can supply a test double for B to reduce the number of test cases.

This way you deal with fewer execution paths and your tests will be simpler. Next, I'm going to show you an example of abusing mocks.