Lecture Notes
November 5, 2020

“Assignment T2: Revised Project Proposal” due tonight
https://courseworks2.columbia.edu/courses/104335/assignments/486956

Every team must discuss their project proposal with their IA mentor *before* the deadline tonight. Make sure your submission states when you met with your IA and what specifically is your response to their feedback.

The IAs reported that they met with some of their teams too briefly and need more time. So we will hold team meetings again in class today, starting around 11am, for some teams. Your IA should have sent you an email if you need to meet today.

Please do not leave when the team meetings start if your mentor still needs to talk to your team!

Working in a team: more on unit testing

Unit testing - tests that individual methods, classes, modules, packages or other “units” do what they are supposed to do to fulfill their role in the design.

Unit tests aim to test a method or class *in isolation from the rest of the program*. They call the method, or a series of methods in the same class, from a test harness rather than from the other code in the rest of the program.

When a unit would normally call other code, use data, or otherwise interact with the environment outside the unit, it instead calls stubs or mocks.

All dependencies are replaced except for basic libraries like string processing. Following CRC design, all the collaborators are replaced by mocks and stubs.

.

Unit tests need to be very fast, so they can run whenever the unit is changed. Real resources are often too slow, but that is not the only reason for using mocks/stubs: When a unit test fails, we know the bug is in that unit, not in some other unit.

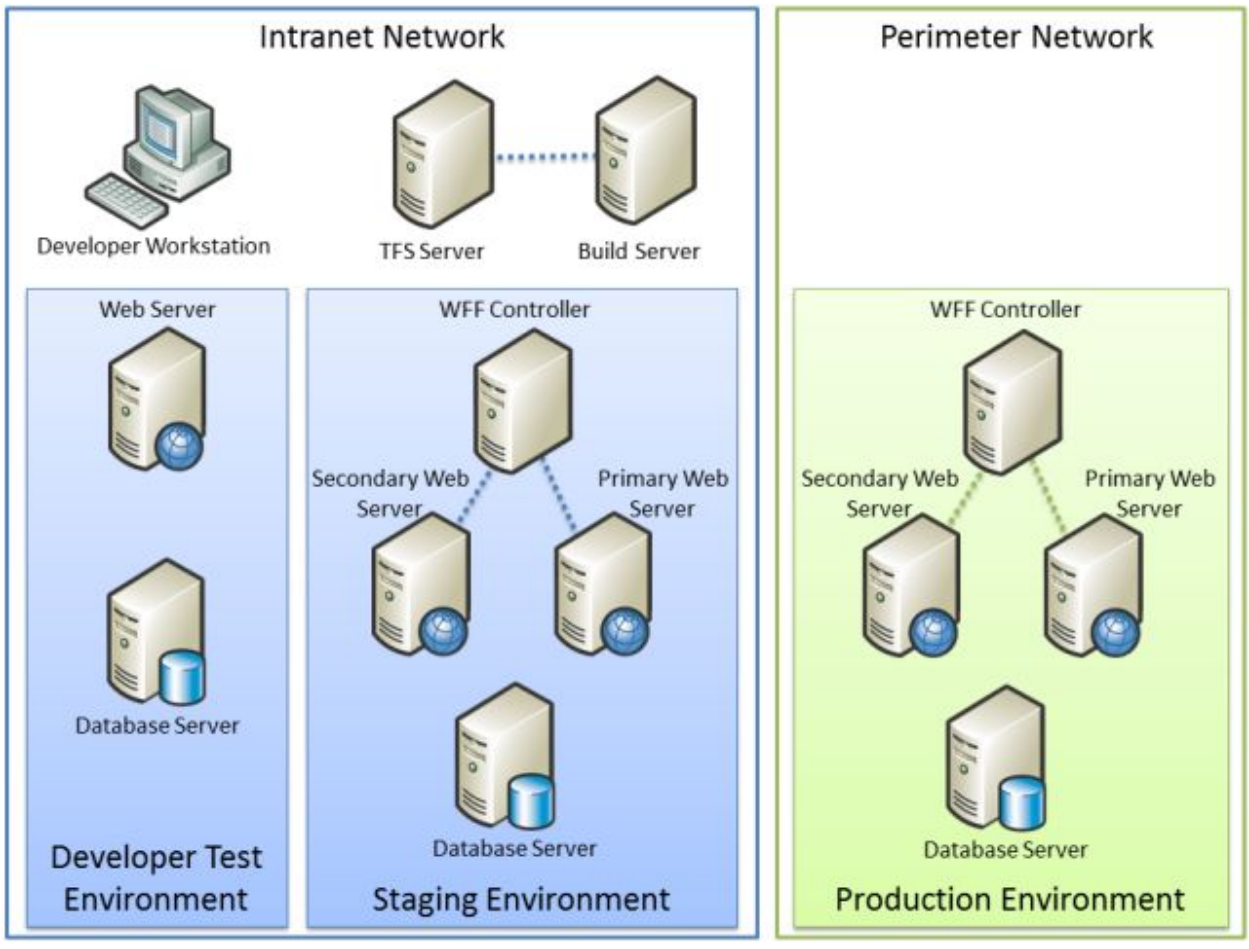
Integration testing - tests boundaries between “units”, to check that the units do what they are supposed to do to fulfill their role in the design.

Usually integrate only two, or a few, units at a time.

When a unit would normally call other code written by our team, it indeed calls our code in the selected other unit(s) intended for integration, but still uses stubs or mocks for all other code.

When an integration test fails, we know the bug is in one of the units being integrated not in some other unit.

We need to use real resources when integration testing with those resources, and in system tests, but we still do not test with the *production* instances of those resources. We do not use production databases, files, or network connections during testing. We do not send emails or tweets to/from user accounts, we do not move real money.



Imagine a newly hired developer using their laptop to test their modified code on the production data for amazon, netflix, facebook, instagram, tiktok, nytimes, ssol, github

What is the difference between a stub and a mock? They are both “test doubles”. Test Double = anything that replaces a production object or service for testing purposes.

Analogous to stunt doubles who perform dangerous action sequences in movies.

Stubs support *state verification*, meaning they provide canned (hardcoded) data. When called during a test, they always return the same value or always do the same thing.

Mocks support *behavior verification*, meaning they actually do something. They may return canned values, but also keep track of the history of how they have been called. Mocks might throw an exception when they receive an unexpected call or don’t receive an expected call. This catches duplicate and out of order method calls.

A stub implements a required interface and delivers *indirect* inputs to the caller when the stub's methods are called. Stubs can replace standard or third-party libraries as well as application code.

```
private class FooStub
implements Foo
{
    public String
bar()
    {
        return "bar";
    }
}
```

```
public class FooCollectionTest
{
    @Test
    public void
it_should_return_joined_bars()
    {
        FooCollection sut = new
FooCollection();
        sut.add(new FooStub);
        sut.add(new FooStub);
        assertEquals("barbar",
sut.joined());
    }
}
```

Stubs are programmed only for the test scope and might provide data to force a specific code path.

Stubs can raise exceptions as well as return values. Suppose we need to simulate a hardware failure or transaction timeout, then the stub always raises the failure or timeout exception.

A dummy object is a special case of a stub that does not even return a value. It's like a stunt double that does no stunts. In movies, sometimes a double doesn't perform anything; they just appear on the screen. For example, the actor's character needs to stand in a crowded place where it would be risky for the real actor to go.

Similarly, a dummy object is passed as a mandatory parameter object. The dummy object is not directly used in the test or in the code under test, but it is required to call the code under test.

```
private class FooDummy
implements Foo
{
    public String bar()
{
        return null;
    }
}
```

```
public class FooCollectionTest
{
    @Test
    public void
it_should_maintain_a_count()
    {
        FooCollection sut = new
FooCollection();
        sut.add(new FooDummy);
        sut.add(new FooDummy);
        assertEquals(2, sut.count());
    }
}
```

Not the same as a null object - which can indeed be used, e.g., in a conditional - if this parameter is null then do x else do y.

A mock generally does more than a stub. It may return hardcoded values for certain calls, but it *remembers* those calls. We can use mocks to verify that all expected actions are performed in the expected order.

```
public class SecurityCentral {
    private final Window window;
    private final Door door;

    public SecurityCentral(Window window, Door door)
{
        this.window = window;
        this.door = door;
    }

    void securityOn() {
        window.close();
        door.close();
    }
}
```

```
public class SecurityCentralTest {
    Window windowMock = mock(Window.class);
    Door doorMock = mock(Door.class);

    @Test
    public void
enabling_security_locks_windows_and_doors() {
        SecurityCentral securityCentral = new
SecurityCentral(windowMock, doorMock);
        securityCentral.securityOn();
        verify(doorMock).close();
        verify(windowMock).close();
    }
```

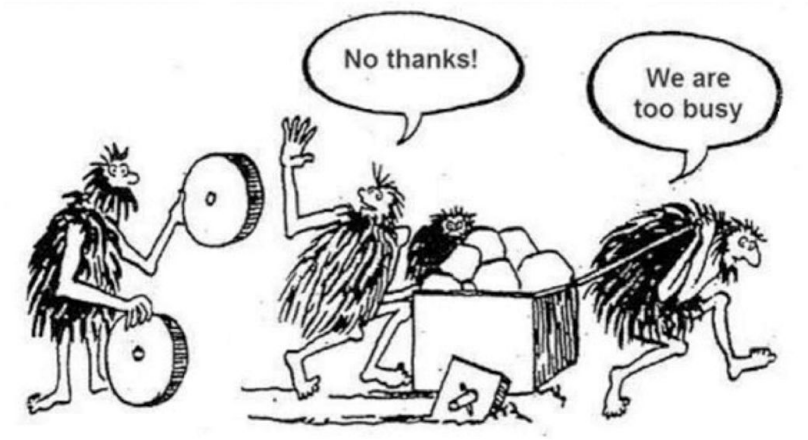
```
}
```

How do we know that SecurityCentral will close the real door and the real window when it needs to? We don't know during unit testing. That is the responsibility of the door and window units.

During integration or system testing, we need to test that security central closes doors and windows.

One way to do this is with a fake object. A fake object is a special case of a mock. It provides a (simplified) working implementation but is not suitable for production use, e.g., an in-memory database, a payments service without real money, a scaled-down network inaccessible by real users.

Another way to do this is with spy objects, another special case of a mock. A spy object can intercept calls to *another* object and log them.



Using a mocking framework is not required for this course this semester (probably will be next year), but unit testing *is* required. Stubs and mocks for unit testing can be created “manually”, but your life will be easier if your team chooses and uses a mocking framework.

C++: gMock, Cgreen

Java: Mockito, PowerMock

Javascript: Sinon, unmock

Python: unittest.mock, pytest-mock

Read more about testing with mocks here.

“Assignment T3: First Iteration” due November 17. https://courseworks2.columbia.edu/courses/104335/assignments/490808

Implement your MVP (minimum viable product). Tell us what user stories you really implemented and what acceptance tests you used to check that the user stories really work. You must use build/package, unit testing, style checking, and static analysis bug finding.

“Assignment T4: Initial Demo” due November 23. Note this is the Monday before Thanksgiving. https://courseworks2.columbia.edu/courses/104335/assignments/491014

“Assignment T2: Revised Project Proposal” due tonight https://courseworks2.columbia.edu/courses/104335/assignments/486956

We will hold meetings now for teams that have not yet worked out an agreed-upon project plan with their IA. If you already had an outside-of-class meeting with your IA, you can use the rest of class time to meet with your team just among yourselves, or you can leave.

Breakout Room Assignments