AlamoCoders

Behavior Driven Development with NUnit

Presented by Joe Ocampo

Today's Agenda

- Contrast TDD with BDD

- Look at some examples of BDD

- Write some code and have fun
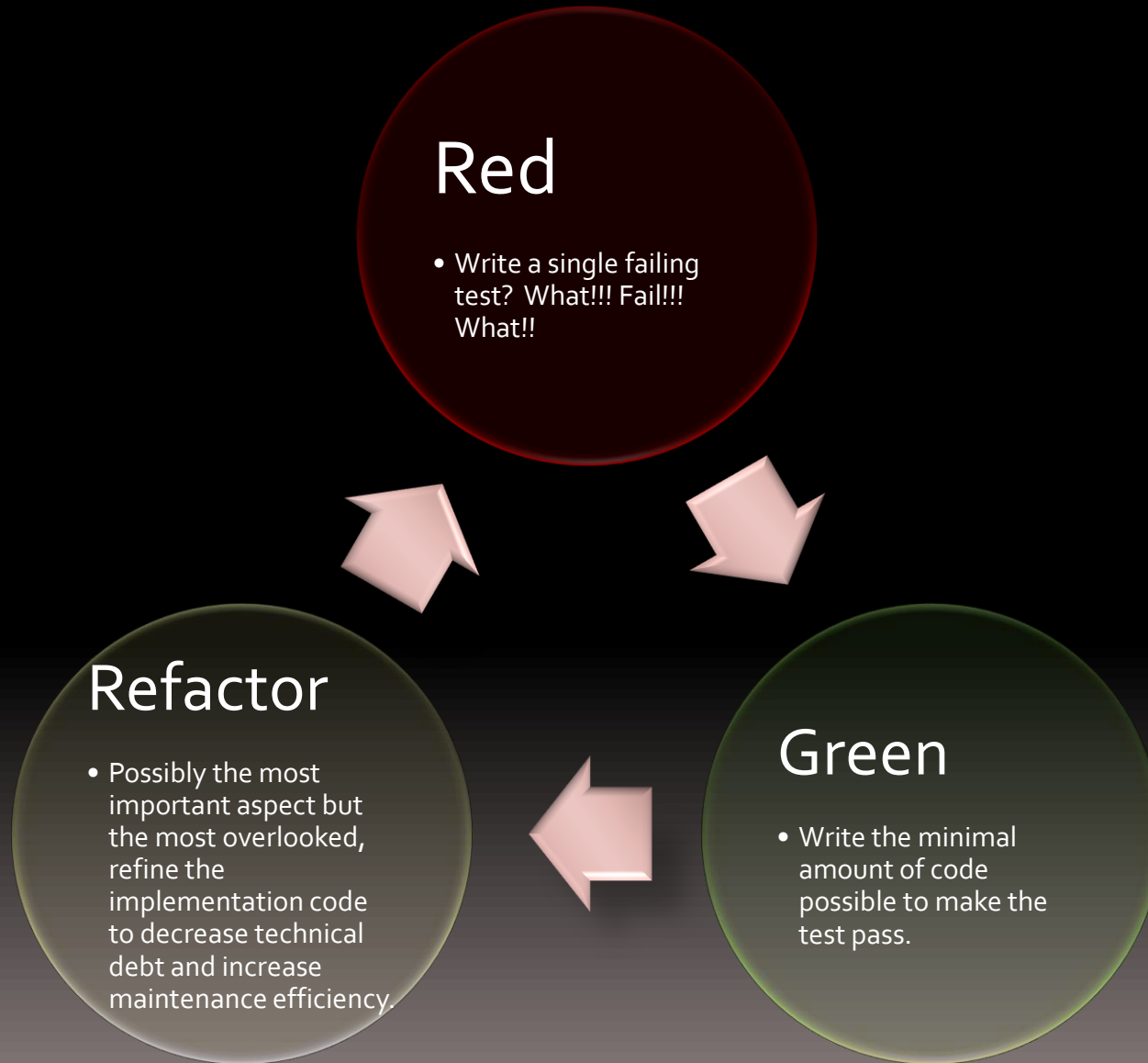
- Answer questions

The issues with Test Driven Development (TDD)

- Where do I start?

- What to Test and What not to test?

- How much do I test in one sitting?

- How do I name my test?

- How do I know why the test failed?

- Test coverage simply for the fact of coverage.

- Eventually loss of understanding.

Enter Behavior Driven Development (BDD)

- Developers like to read about what their software is doing but they hate documentation!

- Lets really make the code the documentation (I am not talking about meta data!)

- Lets make it concise and to the point

- Lets not forget our roots (TDD)

- Everything in the architecture has a reason for being

What did TDD teach us?

Red

- Write a single failing test? What!!! Fail!!! What!!

Refactor

- Possibly the most important aspect but the most overlooked, refine the implementation code to decrease technical debt and increase maintenance efficiency.

Green

- Write the minimal amount of code possible to make the test pass.

Basic TDD Test

```
 [TestFixture]
public class StackTests
{
   [Test]
   public void CreateStack()
   {
     Stack<int> stack = new Stack<int>();

     Assert.IsEmpty(stack);

     stack.Push(1);

     Assert.IsNotEmpty(stack);

     Assert.AreEqual(1, stack.Pop());

   }
}
```

Questions

Should a new stack be empty?

Should a new stack be empty after you push an item on the stack?

When should you pop something off the stack?

Elements of BDD and NUnit

```
namespace AlamoCoders.BDD.Domain.Stack_Specs
{
    [TestFixture]
    public class When_creating_a_stack

    [TestFixture]
    public class When_creating_a_stack_with_one_item

}
```

Questions

Namespace contains the object you are working with "The Given"

Test Fixture Classes dictate the "When" or context of the "Given"

When should you pop something off the stack?

Elements of BDD and NUnit

```csharp
[TestFixture]
public class When_creating_a_stack
{
    private Stack<int> stack;

    [SetUp]
    public void Set_up_context()
    {
        stack = new Stack<int>();
    }
    [Test]
    public void Should_be_empty()
    {
        Assert.IsEmpty(stack, " is not empty");
    }
    [Test]
    public void Should_not_be_empty_after_the_push()
    {
        stack.Push(1);

        Assert.IsNotEmpty(stack, " is empty");
    }
}
```
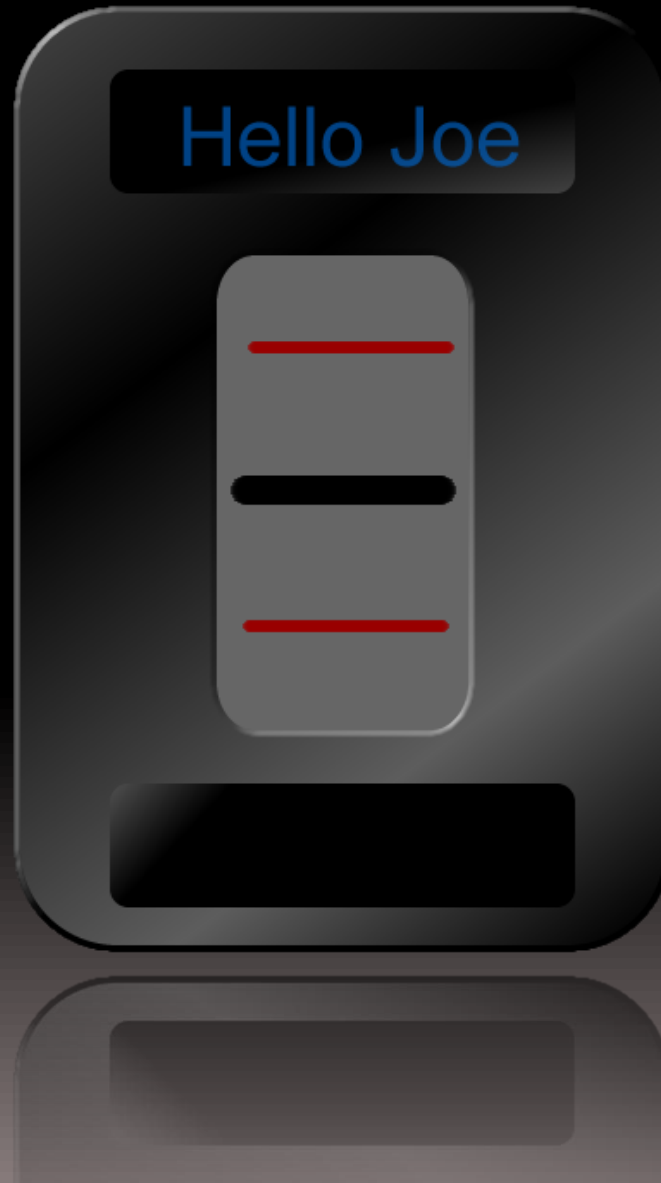
Questions

SetUp attributes are used to set up the context of the test
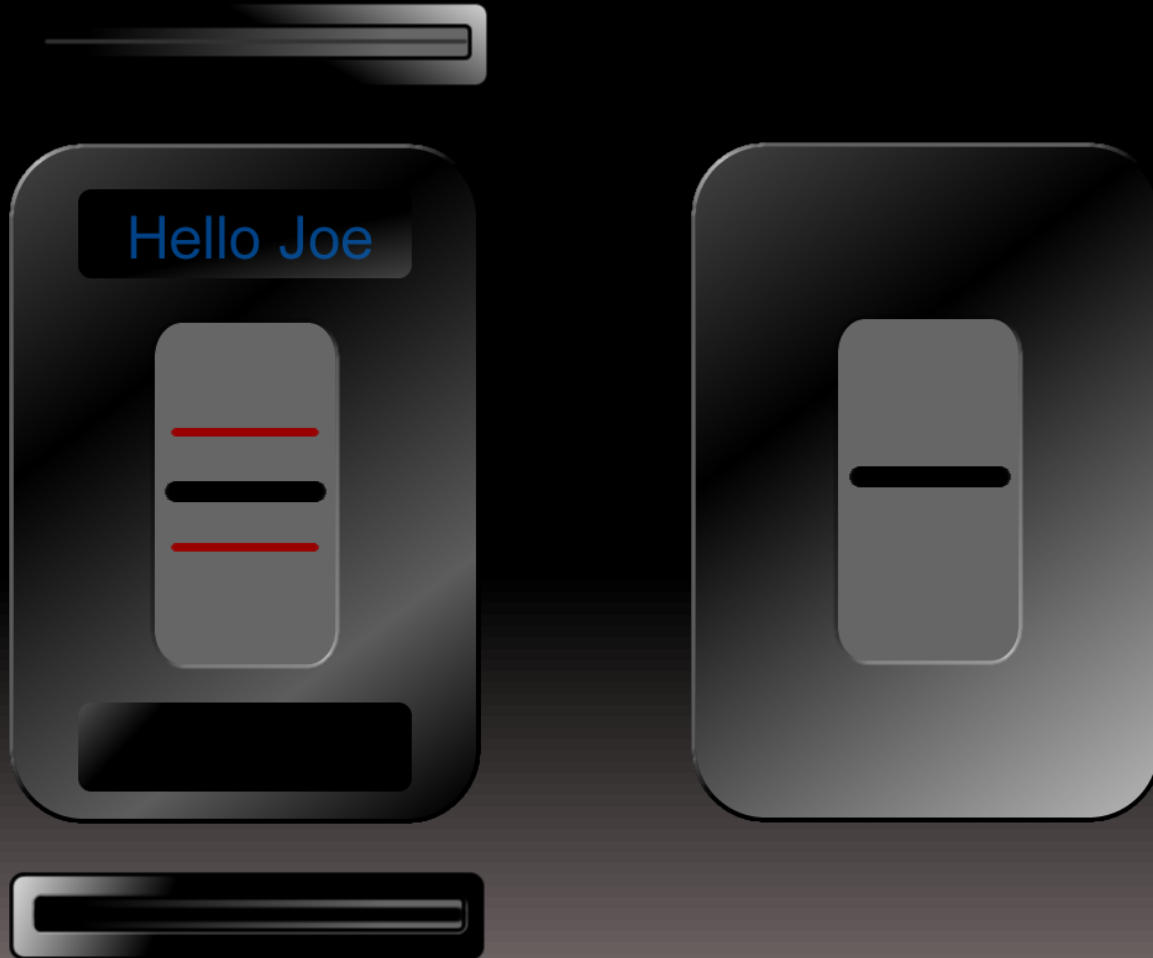
Each test a behavior of the context

And the test runner should read…

- Domain.Stack_Specs. When_creating_a_stack. Should_be_empty

- Domain.Stack_Specs. When_creating_a_stack. Should_not_be_empty_after_the_push

- Domain.Stack_Specs.When_creating_a_stack_ with_one_item. Should_return_the_top_obect_when_popped

Introducing the JWallet

Features of the JWallet

Hello Joe

Features

- Small compact design to mimic the size of an actual wallet
- Made of malleable polymer that will resume it shape when activated
- Can store currency and credit card information.
- Integrated Bluetooth auto syncs with ALL financial software.

Now the issues….

- Hardware is complete on the JWallet but the software has yet to be designed.

- Your mission, should you choose to accept, is to create the software that will interface with the hardware components of the JWallet.

- Please review the following requirement goals.

Requirement Goals

- The user will use the biometric security system (BSS) to authenticate to the wallet

- As bill currency is added to the wallet, the currency slot will read the bill that is being fed to the wallet and store the bill in the wallet and aggregate the total of all the monies in the front panel.

- When the currency slot IR sensors detects a hand gesture and the (BSS) is authenticated the wallet should prompt the user how much money they would like to take out. After the user selects a value the currency slot should retrieve the bill amount.

Rough Model
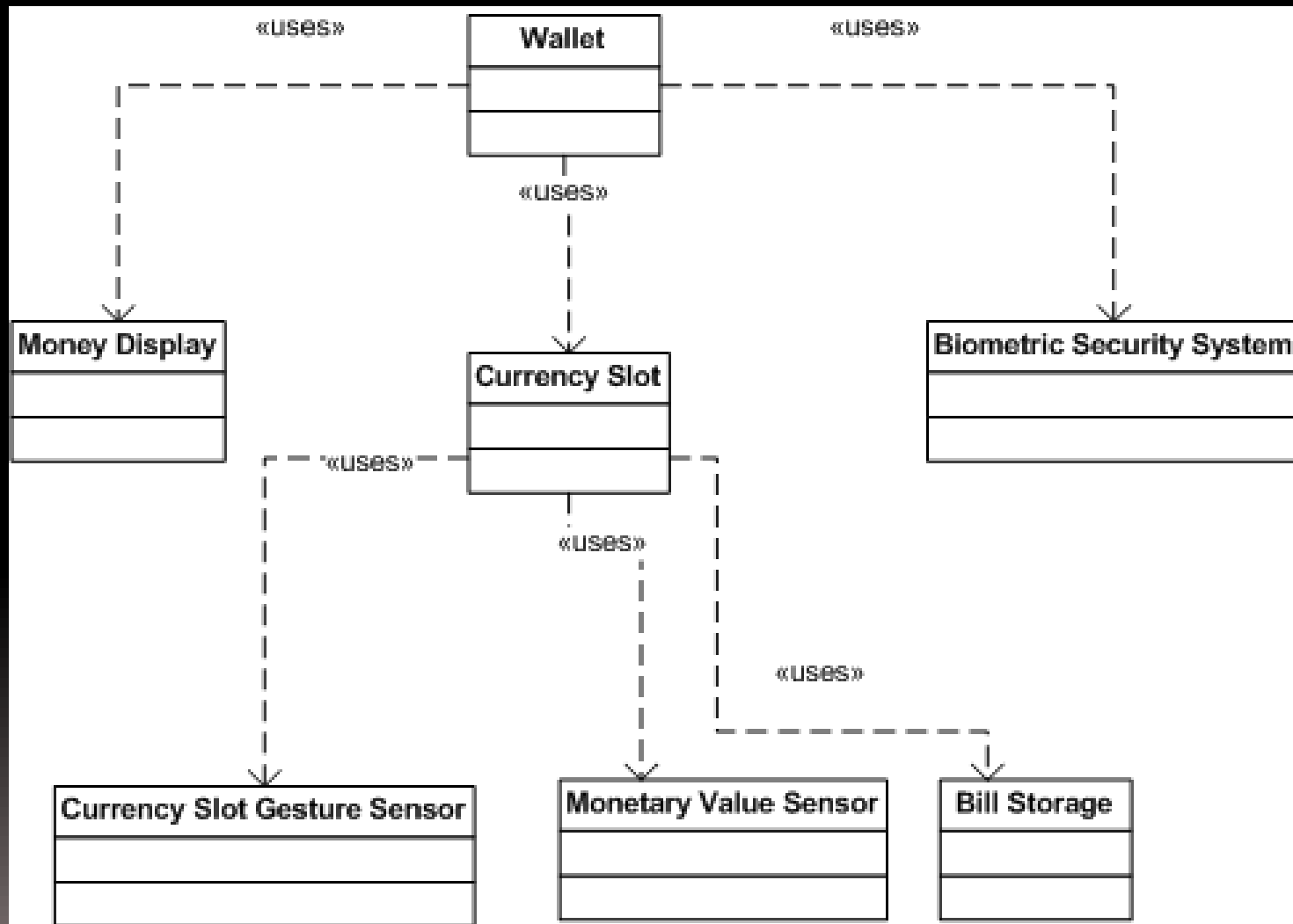
Lets find the "Givens"

- Given I am working with Currency Slot
- When a new bill is being added to the wallet
- Then currency slot should use the monetary sensor to read the value of the bill AND then it should store the money AND then it should total all the monies to display in the Money Display

Guiding principles of BDD

- Work from the Requirement Goals!
- Have a model to work from! Just because xUnit testing is a design practice does not mean that you no longer have to think about an overall architecture! Rough Model, Think YAGNI
- Model, Red, Green, Refactor, Revise, Review
- Apply the Given, When, Then to each object you are creating in order to determine how to write your test.