David Ramsay

~~Stephen R. Covey~~

Remember your training 

- Only test one behaviour at a time 1️⃣
- Only one logical assertion per test 🧪
- Mutually independent tests 🦋
- Don’t mix state and collaboration assertions!

RED-GREEN-REFACTOR

1. Write a failing test 🤦
2. Make it pass 🏎️🏎️
3. Make it ✨beautiful✨

What's a red test? 🤔

It fails!

```
public class BankAccountShould

	[Test]
	public void HaveABalanceOf100OnDeposit()
	{
		BankAccount newAccount = new BankAccount();

		double balance = 0.00;

		Assert.AreEqual(100.00, balance);
	}
```

How can I make it green? 🤔

Make it work!

```
public class BankAccountShould

	[Test]
	public void HaveABalanceOf100OnDeposit()
	{
		BankAccount newAccount = new BankAccount();

		bankAccount.Deposit(100.00);
		double balance = 100.00;

		Assert.AreEqual(100.00, balance);
	}
```

How do I refactor? 🤔

Make it beautiful!

```
public class BankAccountShould
{
      [Test]
      public void HaveABalanceOf100OnDeposit()
      {
            BankAccount newAccount = new BankAccount();

            bankAccount.Deposit(100.00);
            double balance = bankAccount.GetBalance();

            Assert.AreEqual(100.00, balance);
      }
 }
```

```
public class BankAccount
{
      private double _balance = 0.00;
      public void Deposit(double deposit)
      {
            _balance = _balance + deposit;
      }

      public double CurrentBalance()
      {
            return _balance;
      }
}
```

Remember FIRST 🥇

The highly effective Test Driven Developer should implement

- **F**ast ⚡
- **I**solated 🏝️
- **R**epeatable 🪃
- **S**elf validating ✔️
- **T**imely ⏱️

Give meaningful names 🏷️

- Construct class and method names like sentences
  - ❌

```
public class Tests
{
    [Test]
    public void TestBankAccountBalance()
    {
        BankAccount account = new BankAccount();

        var balance = account.CurrentBalance();

        Assert.AreEqual(0.00, balance);
    }
}
```

  - ✔

```
public class BankAccountShould
{
        [Test]
        public void StartWithBalanceOf0()
        {
                BankAccount account = new BankAccount();

                var balance = account.CurrentBalance();

                Assert.AreEqual(0.00, balance);      }
}
```

If it smells like 💩....

- Not testing anything
- Excessive setup
- Too many assertions
- Test too long

- Unclear failing reason

- Checking more than strictly necessary

- Working only on dev machine

- Checking internals

- HIDDEN FUNCTIONALITY BURIED IN THE SETUP

- exception swallowing in test

- Bloated construction impeding test readability

Obsolete test

- Test not belonging logically to the fixture

- Conditional logic test

- Testing or containing irrelevant info

Do’s and don’ts

DO 👍

Keep tests and production code seperate ↔

Model unit tests on the structure of production code 🪞

DON’T 👎

Refactor with failing tests 😡

Use production data or dependencies for tests - only use what you can control 🎮

Thanks for now!