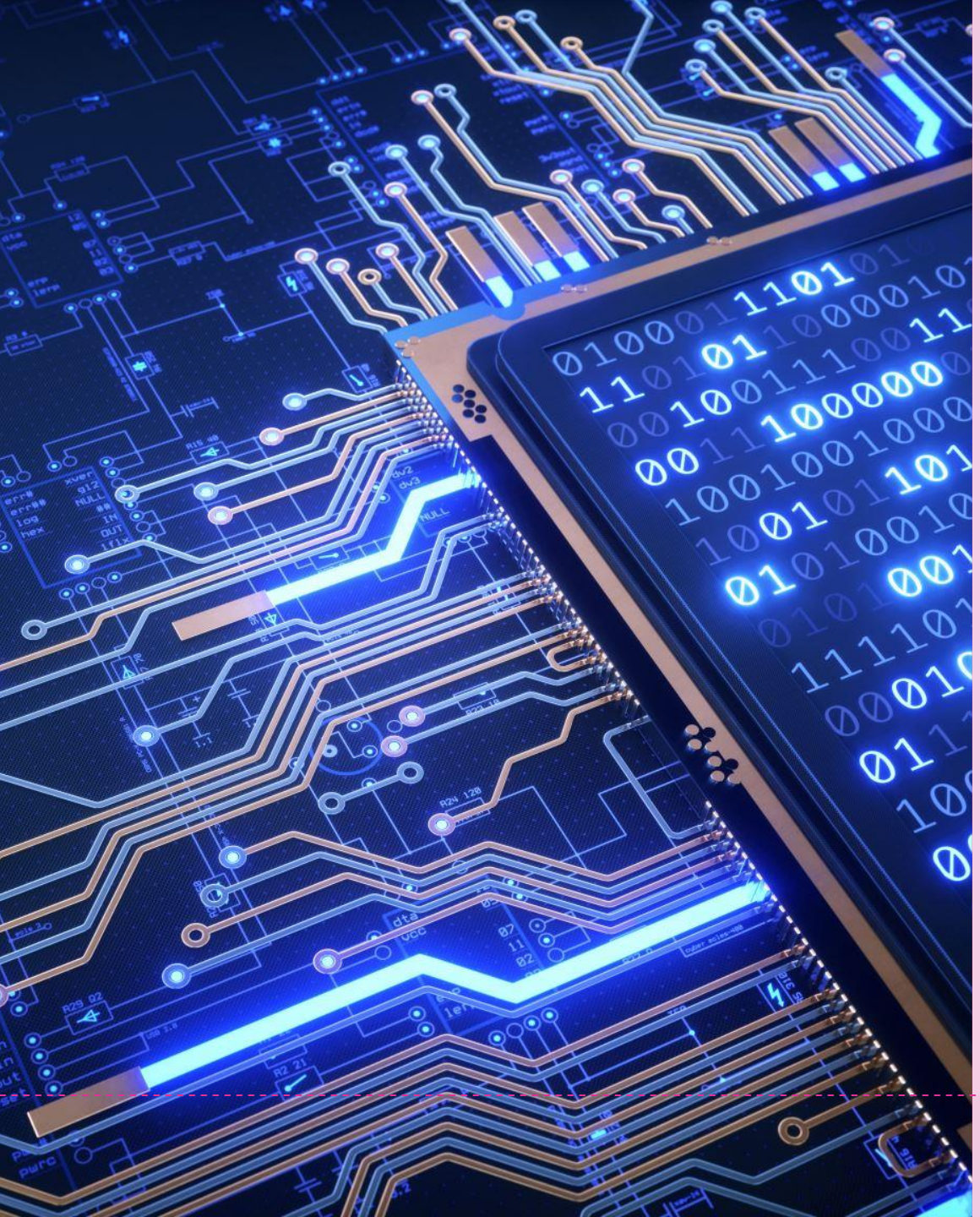

How to use an interface in Testing

By Greg Agnew

Isolation

```
public class RegistrationService
{
        public RegistrationService(
                IFeeCalculator feeCalculator,
                IPaymentProcessor paymentProcessor,
                ISQLRepsoitory sqlRepository,
                IEmailGenerator emailGenerator,
                IEmailSender emailSender)
        {
        }
}
```

```
public interface IEmailSender
{
        void SendMail(MailMessage message);
}
```

```
public class ExpectedSendEmailSender : IEmailSender
{
        private bool sendCalled = false;

        public void SendMail(MailMessage message)
        {
                sendCalled = true;
        }

        public bool SendCalled
        {
                get { return sendCalled; }
        }
}
```

TDD before class design

```
[TestFixture]
public class CommandParserTests
{
    private Mockery mockery;
    [SetUp]
    public void BeforeTest()
    {
        mockery = new Mockery();
    }
    [Test] public void NotifiesListenerOfNewSaleEvent()
    {
        var saleEventListener = mockery.NewMock<ISaleEventListener> ();
        var commandParser = new CommandParser();
        var newSaleCommand = "Command:NewSale";
        commandParser.Parse(newSaleCommand);
    }
}
```

Moq and NUnit

- Moq is a library that allows mocking of various things including interfaces.
- Nunit is an open-source testing framework similar to Xunit.

```
1  [TestFixture]
2  public class FooTest
3  {
4      Foo subject;
5      Mock myInterfaceMock;
6
7      [SetUp]
8      public void SetUp()
9      {
10         myInterfaceMock = new Mock();
11         subject = new Foo();
12     }
13 }
```

Easier Test Duplication

```
[TestClass]
public abstract class StringSearcherTestBase
{
    /// <summary>
    /// Override this method to implement the tests
    /// </summary>
    /// <returns></returns>
    public abstract IStringSearcher GetStringSearcherInstance();

    [TestMethod]
    public void BasicTest()
    {
        IStringSearcher searcher = GetStringSearcherInstance();
        List<int> indexes = searcher.SearchString(
            "Hello. Welcome to unit testing interfaces",
            "test").ToList();

        Assert.AreEqual(1, indexes.Count);
        Assert.AreEqual(23, indexes[0]);
    }

    [TestMethod]
    public void NegativeTest()
    {
        IStringSearcher searcher = GetStringSearcherInstance();
        var indexes = searcher.SearchString(
            "Hello. Welcome to unit testing interfaces",
            "uint").ToList();

        Assert.AreEqual(0, indexes.Count);
    }
```

```
[TestClass]
public class StringSearcherBoyerMoore_Tests : StringSearcherTestBase
{
    public override IStringSearcher GetStringSearcherInstance()
    {
        return new StringSearcherBoyerMoore();
    }
}
```

Or Maybe Not…

- https://softwareengineering.stackexchange.com/questions/159813/do-i-need-to-use-an-interface-when-only-one-class-will-ever-implement-it/159821#159821

Resources

- https://spin.atomicobject.com/2017/08/07/intro-mocking-moq/
- https://www.lambdatest.com/blog/nunit-vs-xunit-vs-mstest/
- https://docs.microsoft.com/en-us/archive/msdn-magazine/2009/june/using-mock-objects-and-tdd-to-design-role-based-objects
- https://www.codeproject.com/Tips/609259/Unit-Testing-Interfaces-in-NET
- https://visualstudiomagazine.com/articles/2010/01/01/interface-based-programming.aspx
- https://softwareengineering.stackexchange.com/questions/159813/do-i-need-to-use-an-interface-when-only-one-class-will-ever-implement-it/159821#159821