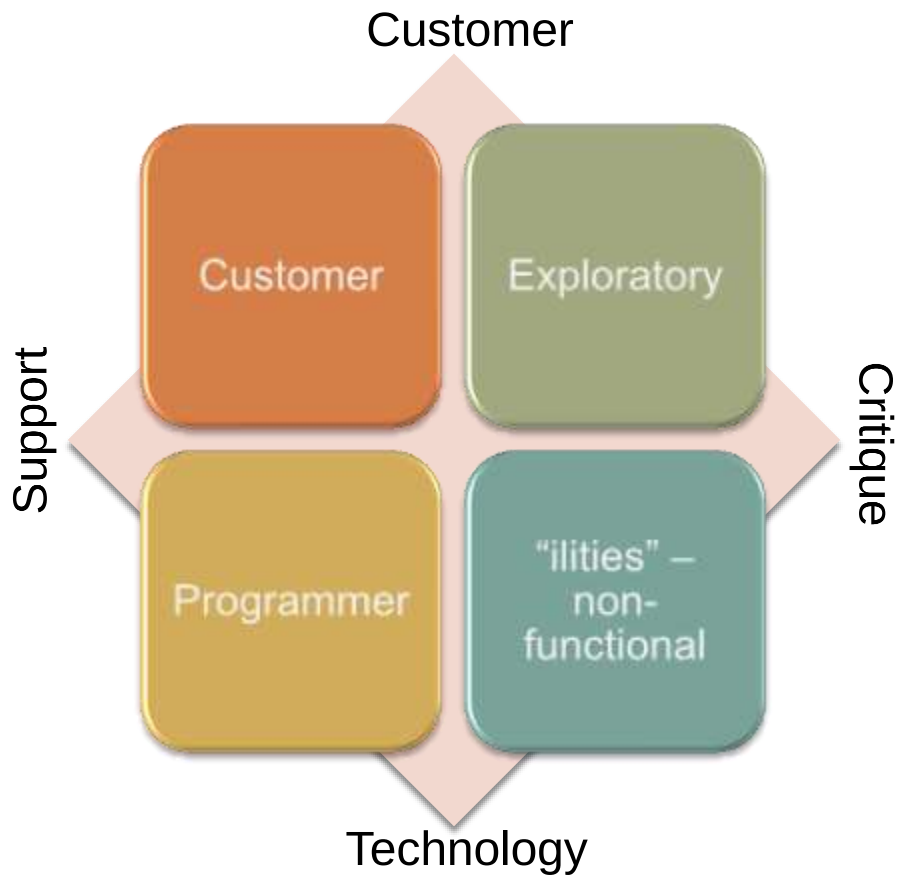
LESSONS LEARNED IN PROGRAMMER TESTING

PATTERNS AND IDIOMS

James Newkirk and Brad Wilson

What is Programmer Testing?

- Brian Marick
  - http://www.testing.com



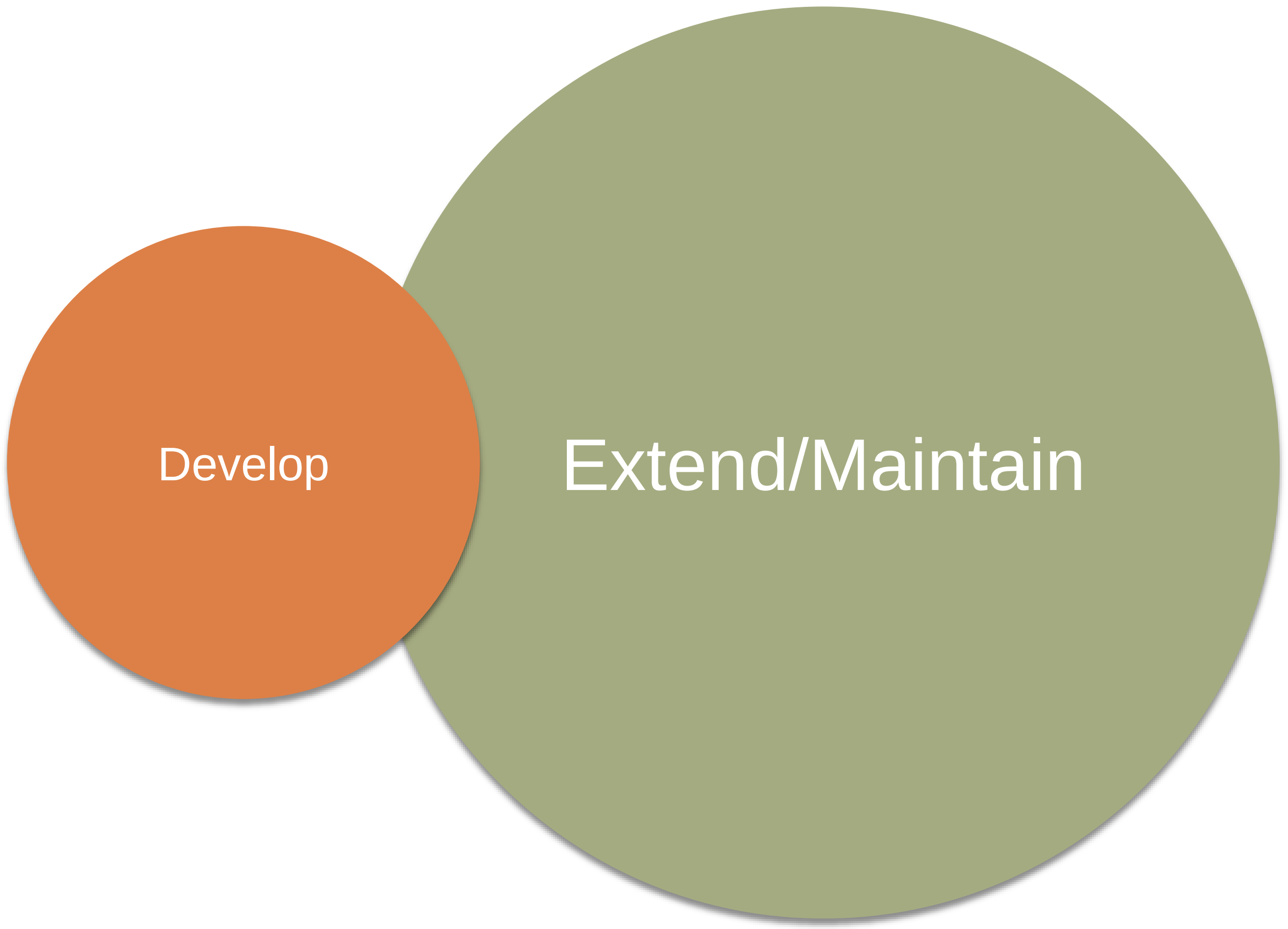
Do you do programmer testing?

- How many of you have been doing programmer testing for 5 years or more?
- 4 years
- 3 years
- 2 years
- 1 year

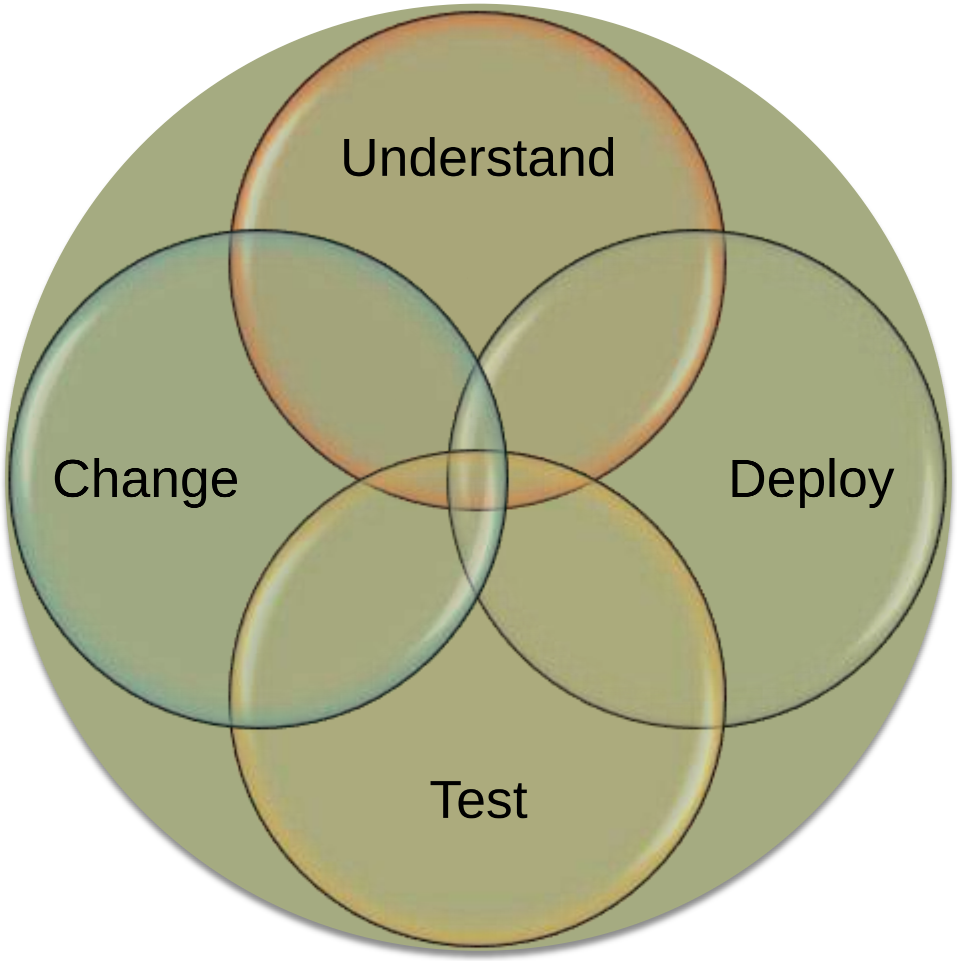
Why do programmer testing?

- *“There is no such thing as done. Much more investment will be spent modifying programs than developing them initially”* [Beck]
- *“Programs are read more often than they are written”* [Beck]
- *“Readers need to understand programs in detail and concept”* [Beck]

Total Development Cost



Extend/Maintain Cost [Beck]



I might break something

Where do I start?

Just Do It!

Lesson #1

Write tests using the 3A pattern

3A Pattern

- Attributed to Bill Wake (http://xp123.com)
  - Arrange – Setup the test harness
  - Act – Run the test
  - Assert – Check the results
- Let’s look at an example!

A Typical Test

```
[Fact]
public void TopDoesNotChangeTheStateOfTheStack()
{
    Stack<string> stack = new Stack<string>();
    stack.Push("42");


    string element = stack.Top;


    Assert.False(stack.IsEmpty);
}
```

3A Pattern

```
[Fact]
public void TopDoesNotChangeTheStateOfTheStack()
{
    Stack<string> stack = new Stack<string>();
    stack.Push("42");                                  Arrange


    string element = stack.Top;


    Assert.False(stack.IsEmpty);
}
```

3A Pattern

```
[Fact]
public void TopDoesNotChangeTheStateOfTheStack()
{
    Stack<string> stack = new Stack<string>();
    stack.Push("42");


    string element = stack.Top;                              Act


    Assert.False(stack.IsEmpty);
}
```

3A Pattern

```
[Fact]
public void TopDoesNotChangeTheStateOfTheStack()
{
    Stack<string> stack = new Stack<string>();
    stack.Push("42");


    string element = stack.Top;


    Assert.False(stack.IsEmpty);                    Assert
}
```

3A Summary

- Benefits
  - Readability
  - Consistency
- Liabilities
  - More Verbose
  - Might need to introduce local variables
- Related Issues
  - One Assert per Test?

Lesson #2

Keep Your Tests Close

Keep Your Tests Close

- Benefits
  - Tests are equivalent to production code
  - Solves visibility problems
- Liabilities
  - Should you ship your tests?
  - If No, how do you separate the tests from the code when you release?

Lesson #3

ExpectedException leads to uncertainty

ExpectedException Violates 3A

```
[Test]
[ExpectedException(typeof(InvalidOperationException))]
public void PopEmptyStack()
{
    Stack<string> stack = new Stack<string>();

    stack.Pop();
}
```

Record the Exception instead

```
[Fact]
public void PopEmptyStack()
{
    Stack<string> stack = new Stack<string>();

    Exception ex = Record.Exception(() => stack.Pop());

    Assert.IsType<InvalidOperationException>(ex);
}
```

Use Assert.Throws - .NET 2.0

```
[Fact]
public void PopEmptyStack()
{
    Stack<string> stack = new Stack<string>();

    Assert.Throws<InvalidOperationException>(
        delegate
        {
            stack.Pop();
        });
}
```

Use Assert.Throws - .NET 3.5

```
[Fact]
public void PopEmptyStack()
{
    Stack<string> stack = new Stack<string>();

    Assert.Throws<InvalidOperationException>(
        () => stack.Pop());
}
```

More ExpectedException Problems

```
[Test, ExpectedException(typeof(ArgumentException))]
public void DepositThrowsArgumentExceptionWhenZero()
{
    CheckingAccount account = new CheckingAccount(0.00);

    account.Deposit(0.00);
}
```

```
public CheckingAccount(double balance)
{
    if (balance == 0) throw new ArgumentException("...");
}

public void Deposit(double amount)
{
    if(amount == 0) throw new ArgumentException("...");
}
```

Use Assert.Throws

```
[Fact]
public void DepositThrowsArgumentExceptionWhenZero()
{
    CheckingAccount account = new CheckingAccount(150.00);

    Assert.Throws<ArgumentException>(
        () => account.Deposit(0));
}
```

```
public void Deposit(Decimal amount)
{
    if(amount == 0) throw new ArgumentException("...");

    // the rest of the implementation
}
```

Improved Control Flow

```
[Fact]
public void PopEmptyStack()
{
    Stack<string> stack = new Stack<string>();

    Exception ex = Record.Exception(() => stack.Pop());

    Assert.IsType<InvalidOperationException>(ex);
    Assert.Equal("Stack empty.", ex.Message);
}
```

Use Alternatives to ExpectedException

- Benefits
  - Readability (these tests look like all the rest)
  - Identify and isolate the code where you are expecting the exception
  - Improved control flow
- Liabilities
  - Act and Assert are together in Assert.Throws
  - Anonymous delegate syntax in .NET Framework 2.0 is not great for readability

Lesson #4

Small Fixtures

Small Fixtures

- Benefits
  - Smaller more focused test classes
  - Class contains nested classes
- Liabilities
  - Potential code duplication
  - Issues with test runners
- Related Issues
  - Do you need SetUp and TearDown?

Lesson #5

Don’t Use SetUp or TearDown

Don’t Use SetUp orTearDown

- Benefits
  - Readability
  - Test isolation
- Liabilities
  - Duplicated initialization code
- Related Issues
  - Small Fixtures

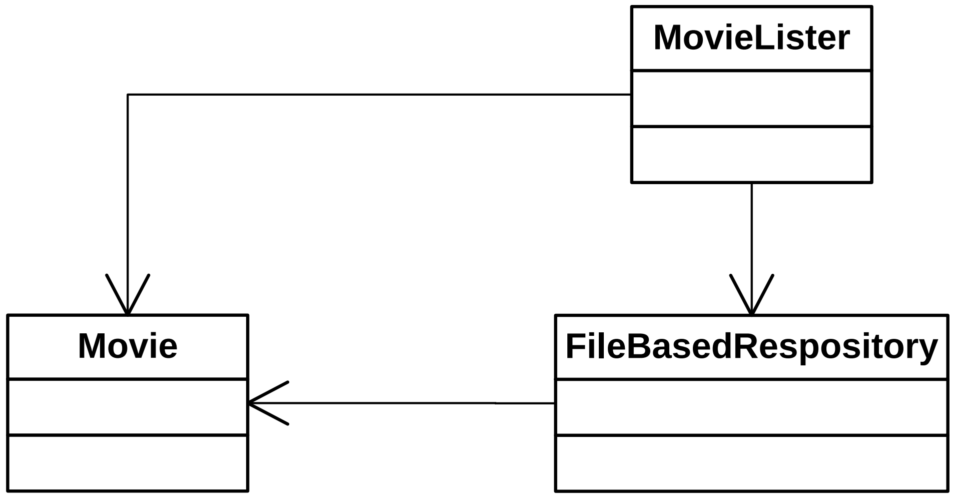
Lesson #6

Improve Testability with Inversion of Control

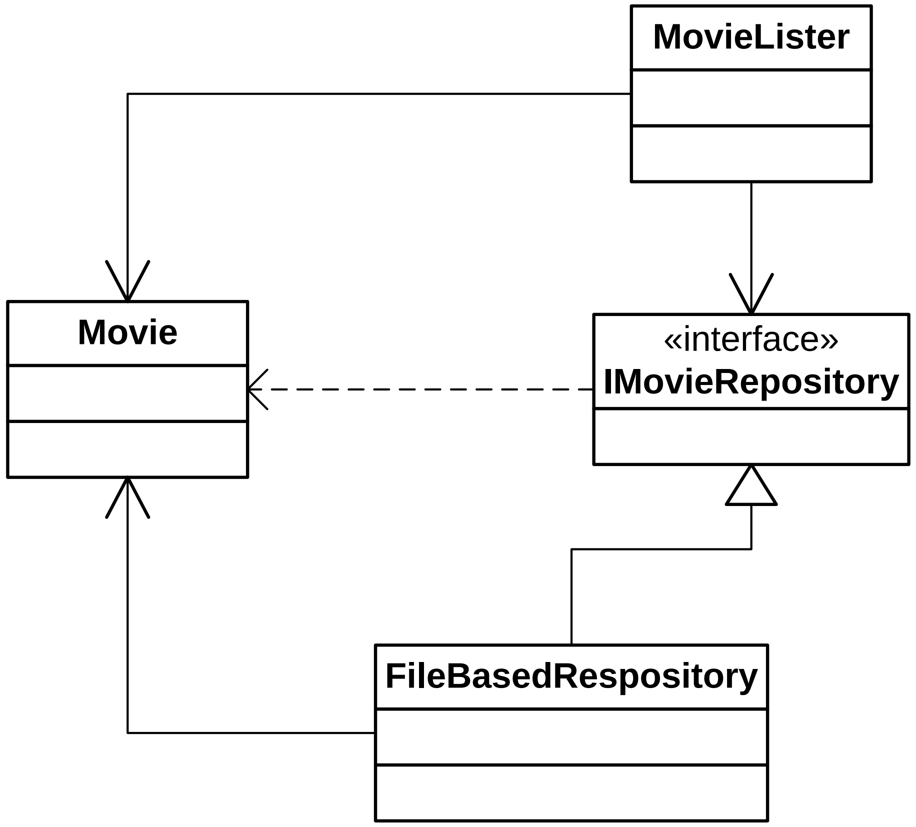
Inversion of Control (IoC) Pattern

- Article: **http://shrinkster.com/wkm**
- Dependency Injection
  - Constructor Injection
  - Setter Injection
- Let’s look at an example from the article!

Before



After


MovieLister
Movie
«interface»
IMovieRepository
FileBasedRespository

Improve Testability with IoC

- Benefits
  - Better test isolation
  - Decoupled class implementation
- Liabilities
  - Decreases encapsulation
  - Interface explosion
- Related Issues
  - Dependency injection frameworks are overkill for most applications

Summary

- Just Do It!
- Lesson #1 – Write Tests using the 3A Pattern
- Lesson #2 – Keep your tests Close
- Lesson #3 – Use Alternatives to ExpectedException
- Lesson #4 – Small Fixtures
- Lesson #5 – Don’t use SetUp or TearDown
- Lesson #6 – Improve Testability with Dependency Injection

Tools

- xUnit.net – http://codeplex.com/xunit
- Nunit – http://nunit.org
- MbUnit – http://mbunit.com
- Visual Studio 2008 - http://msdn2.microsoft.com/enus/vstudio/default.aspx

Blogs

- Brian Button http://www.agileprogrammer.com/oneagilecoder
- Brian Marick http://www.testing.com/cgi-bin/blog
- Peter Provost http://peterprovost.org

Books

- [Beck] Implementation Patterns by Kent Beck, Addison-Wesley, 2008
- xUnit Test Patterns by Gerard Meszaros, Addison-Wesley, 2007
- Refactoring to Patterns by Joshua Kerievsky, Addison-Wesley, 2005

Contact Information

- James Newkirk
  - Email: jamesnew@microsoft.com
  - Blog: http://jamesnewkirk.typepad.com
- Brad Wilson
  - Email: bradwils@microsoft.com
  - Blog: http://bradwilson.typepad.com
  - Twitter: http://twitter.com/bradwilson

Questions