Strategies and Techniques for Testing Code

Course Outline

Software development challenges

What is test-driven development?

Different ways of testing applications

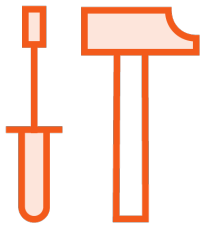
Test-driven development in action

Strategies/Techniques for testing code

Test-driven development gotchas

Strategies and Techniques

Dependency Injection



Test Doubles

Best Practices

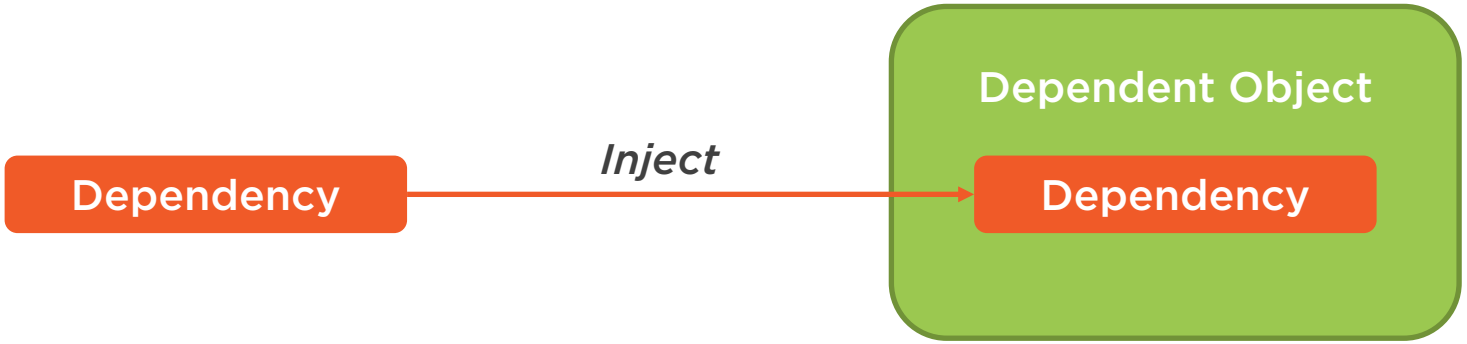
Dependency Injection

What Is Dependency Injection?

In software engineering, dependency injection is a technique whereby one object supplies the dependencies of another object.

- A *dependency* is an object that can be used.

- An *injection* is the passing of a dependency to a dependent object that would use it.



Wikipedia - https://en.wikipedia.org/wiki/Dependency_injection

The Testing Problem

```
public class Greeter
{
  private NameService nameService;

  public Greeter()
  {
    this.nameService = new NameService();
  }

  public string SayHello()
  {
    return $"Hello, {this.nameService.GetName()}";
  }
}
```

Dependency

What if we want this to fail?

Three Types of Dependency Injection

Constructor Injection

Providing dependencies through a class constructor.

Property/Setter Injection

Using a property or setter method to inject a dependency.

Interface Injection

Client defines interface that describes how dependencies are injected into it.

Constructor Injection

```
public interface INameService
{
  string GetName();
}

public class Greeter
{
  private INameService nameService;
  public Greeter(INameService nameService)
  {
    this.nameService = nameService;
  }

  public string SayHello()
  {
    var txt = “Hello, ”;
    txt += this.nameService.GetName();
    return txt;
  }
}
```

Test

```
public class TestNameService : INameService
{
  private string Name { get; set; }
  public TestNameService(string name)
  {
    this.Name = name;
  }

  public string GetName()
  {
    return this.Name;
  }
}
```

```
var expected = “Hello, Jason”;
var service = new TestNameService(“Jason”);
var greeter = new Greeter(service);
Assert.Equal(expected, greeter.SayHello());
```

Property/Setter Injection

```
public interface INameService
{
  string GetName();
}

public class Greeter
{
  public INameService NameService
  {
    get; set;
  }

  public string SayHello()
  {
    var txt = “Hello, ”;
    txt += this.nameService.GetName();
    return txt;
  }
}
```

Test

```
var expected = “Hello, Jason”;
var service = new TestNameService(“Jason”);

var greeter = new Greeter();
greeter.NameService = service;
Assert.Equal(expected, greeter.SayHello());
```

Interface Injection

```
public interface INameService
{
  string GetName();
}

public interface INameServiceInjector
{
  INameService NameService { get; set; }
}

public class Greeter : INameServiceInjector
{
  public INameService NameService
  {
    get; set;
  }

  public string SayHello()
  {
    var txt = “Hello, ”;
    txt += this.NameService.GetName();
    return txt;
  }
}
```

Test

```
var expected = “Hello, Jason”;
var service = new TestNameService(“Jason”);

var greeter = new Greeter();
greeter.NameService = service;
Assert.Equal(expected, greeter.SayHello());
```

Is Dependency Injection Only for Object-Oriented Programming?

```
function getName() {
  return "Jason";
}

function greet() {
  return `Hello, ${getName()}`;
}

greet();
//=> Hello, Jason
```

```
function greet(getName) {
  return `Hello, ${getName()}`;
}

greet(() => {
  return "Jason";
});
//=> Hello, Jason
```

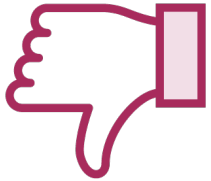
Higher-Order Functions

Trade-offs of Dependency Injection

Self-contained Code

Dependency-injected Code

Easier to Understand



Harder to Test

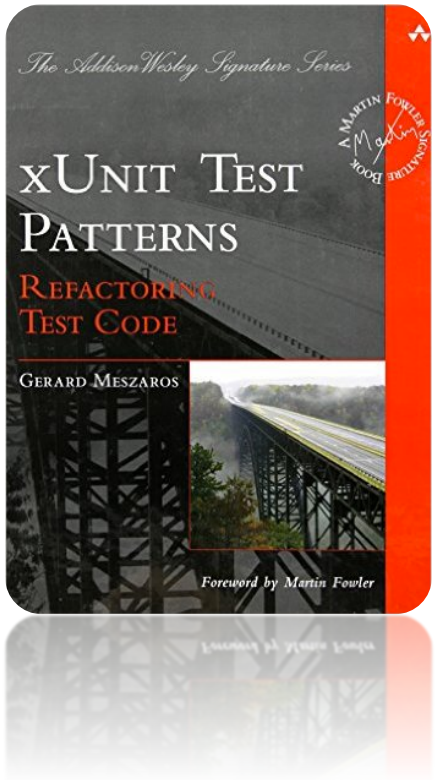
Harder to Understand

Easier to Test

Test Doubles

Test Doubles

Gerard Meszaros



***Test Double** is a "generic term for any kind of pretend object used in place of a real object for testing purposes."*

Martin Fowler
martinfowler.com/articles/mocksArentStubs.html

Code

Dependency Injection

```
public interface INameService
{
  string GetName();
}

public class Greeter
{
  private INameService nameService;
  public Greeter(INameService nameService)
  {
    this.nameService = nameService;
  }

  public string SayHello()
  {
    var txt = “Hello, ”;
    txt += this.nameService.GetName();
    return txt;
  }
}
```

Test

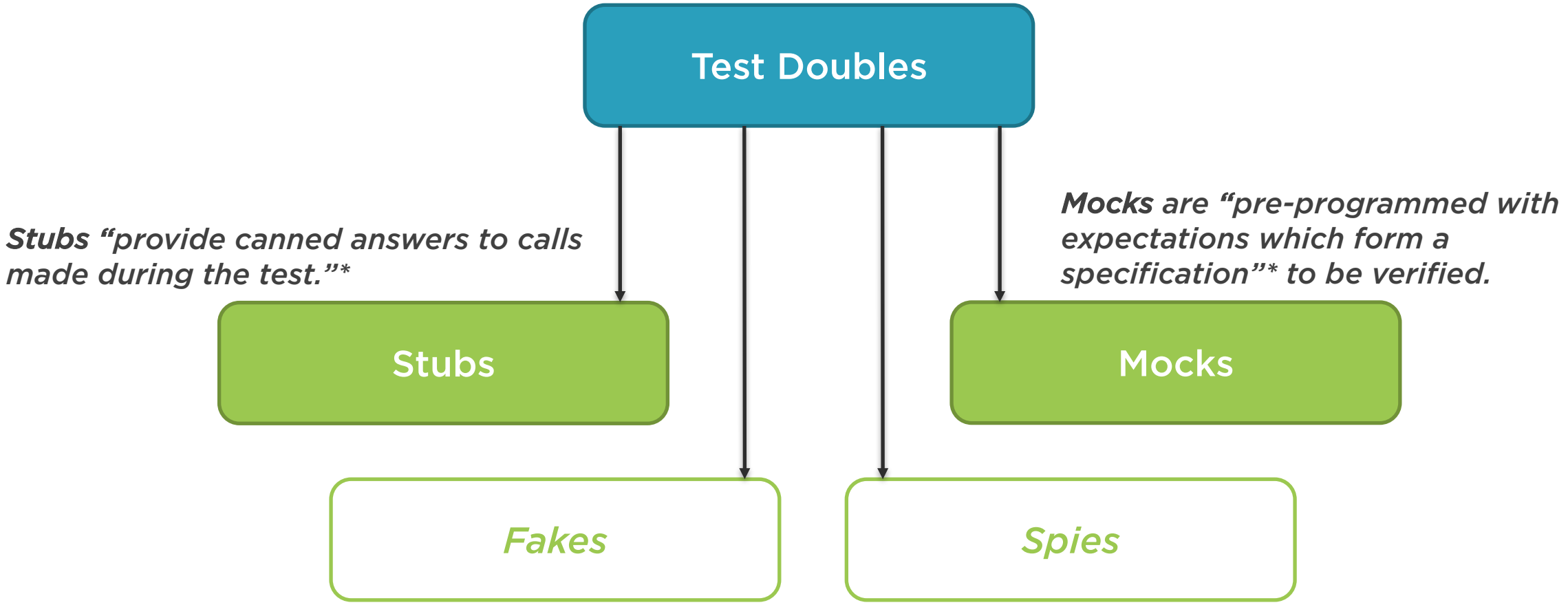
Test Double

```
public class TestNameService : INameService
{
  private string Name { get; set; }
  public TestNameService(string name)
  {
    this.Name = name;
  }

  public string GetName()
  {
    return this.Name;
  }
}
```

```
var expected = “Hello, Jason”;
var service = new TestNameService(“Jason”);
var greeter = new Greeter(service);
Assert.Equal(expected, greeter.SayHello());
```

Types of Test Doubles



* martinfowler.com/articles/mocksArentStubs.html

Stub

```
public class TestNameService : INameService
{
  private string Name { get; set; }
  public TestNameService(string name)
  {
    this.Name = name;
  }

  public string GetName()
  {
    return this.Name;
  }
}
```

```
var expected = “Hello, Jason”;
var service = new TestNameService(“Jason”);
var greeter = new Greeter(service);
Assert.Equal(expected, greeter.SayHello());
```

Mock *(moq4)*

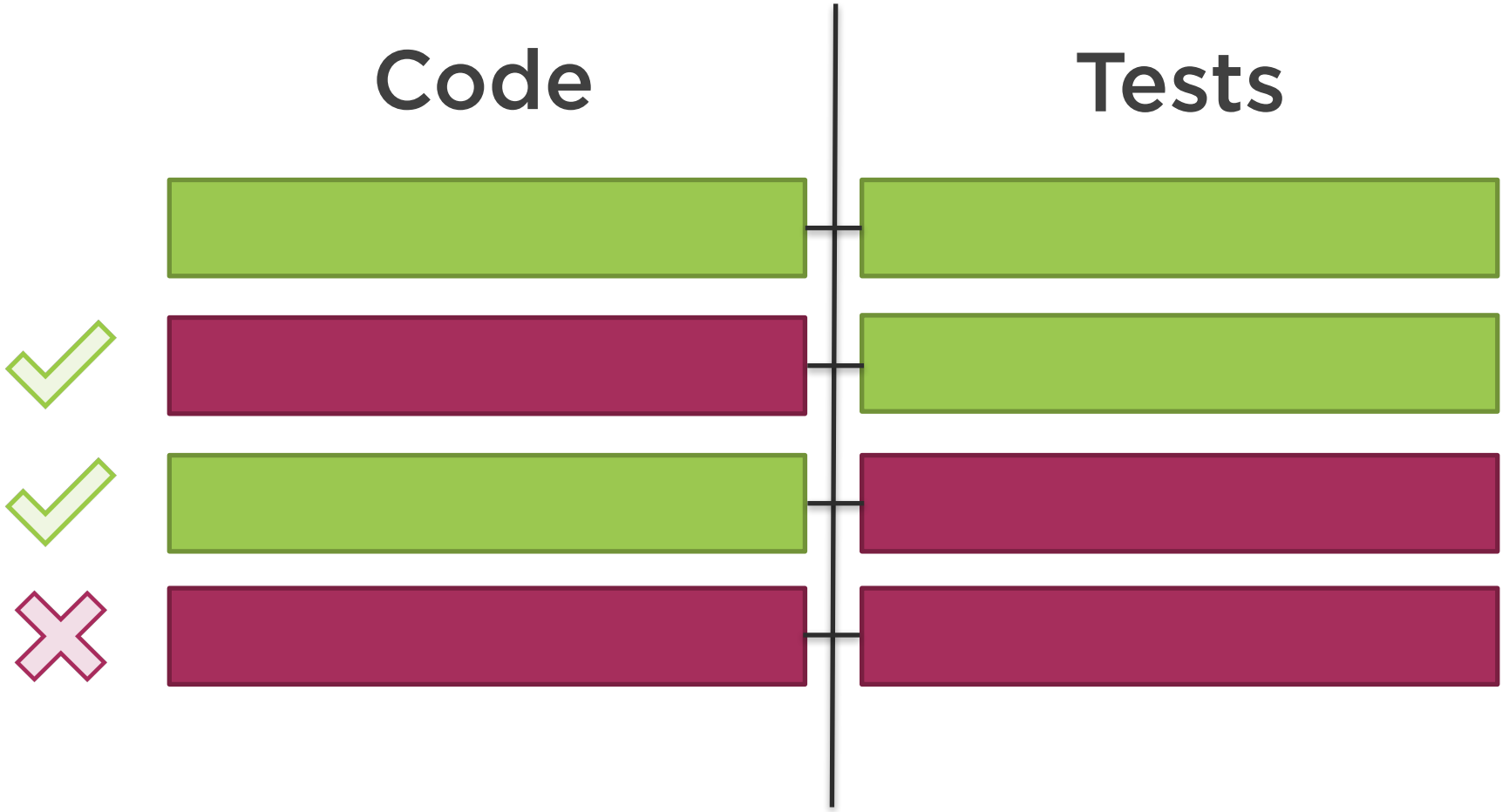
```
var mock = new Mock<INameService>();
mock.Setup(ns => ns.GetName()).Returns(“Jason”);
```

```
var expected = “Hello, Jason”;
var greeter = new Greeter(mock);
Assert.Equal(expected, greeter.SayHello());
```

```
mock.Verify(ns => ns.GetName(),
  Times.AtLeastOnce());
```

More Brittle

Brittle Code



Testing Best Practices

Treat Test Code Like Production Code

Write readable and maintainable test code

Address both positive and negative test cases

Separate common set-up and teardown logic

Focus Only on Necessary Values and Results

Only assert on values required to verify the test is passing

Review Tests and Test Practices with Team

Effective techniques

Catching bad habits

Common challenges

Leave the Code Better Than You Found It

Summary

Software development challenges

What is Test-Driven Development?

Different ways of testing applications

Test-Driven Development in action

Strategies/Techniques for testing code

Test-Driven Development gotchas