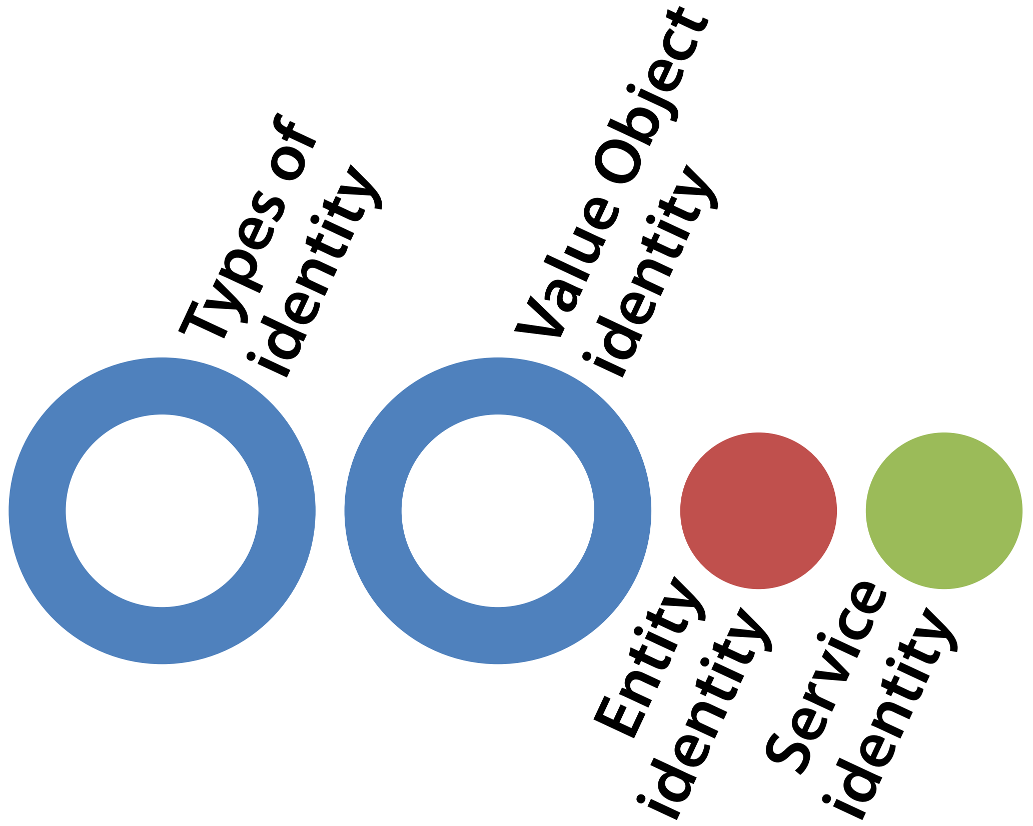
Advanced Unit Testing
Identity

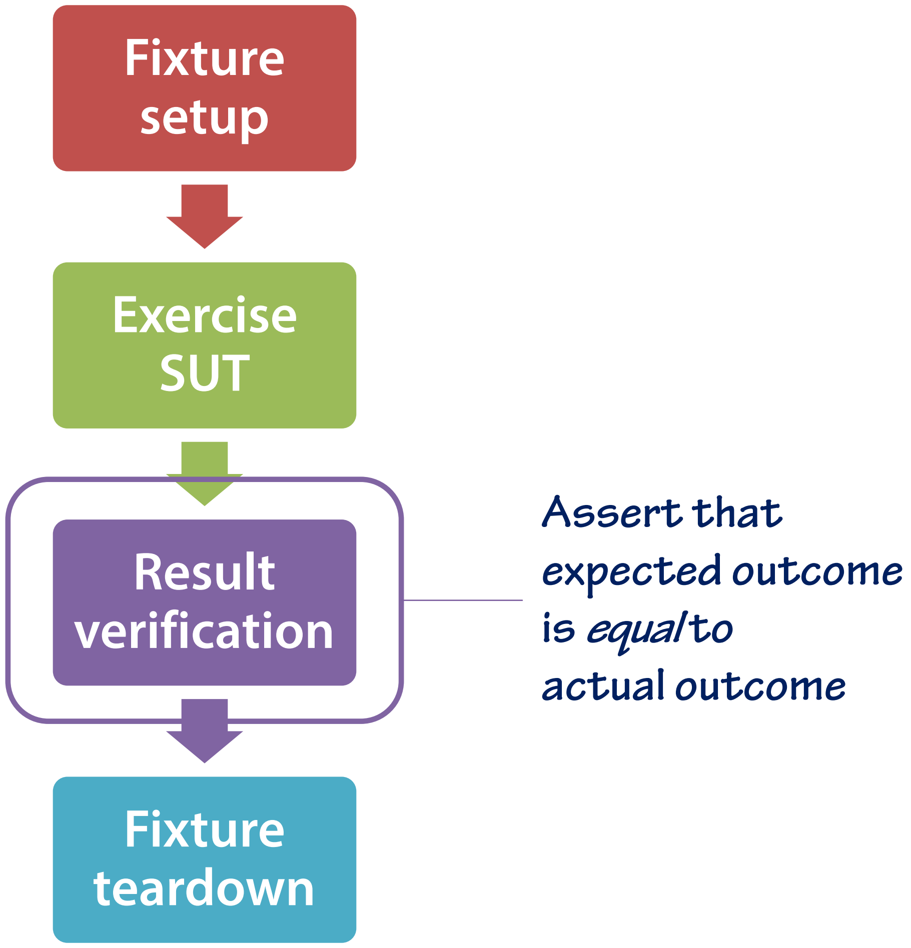
Mark Seemann

http://blog.ploeh.dk

Overview


Types of identity
Value Object identity
Entity identity
Service identity

Four-Phase Test



Assertion Roulette

```
[Fact]
public void UseBasketPipelineOnExpensiveBasket()
{
    // Fixture setup
    var basket = new Basket(
        new BasketItem("Chocolate", 50, 3),
        new BasketItem("Gruyère", 45.5m, 1),
        new BasketItem("Barolo", 250, 2));
    CompositePipe<Basket> pipeline = new BasketPipeline();
    // Exercise system
    var actual = pipeline.Pipe(basket);
    // Verify outcome
    var bi1 = Assert.IsAssignableFrom<BasketItem>(
        actual.ElementAt(0));
    Assert.Equal("Chocolate", bi1.Name);
    Assert.Equal(50, bi1.UnitPrice);
    Assert.Equal(3, bi1.Quantity);

    var bi2 = Assert.IsAssignableFrom<BasketItem>(
        actual.ElementAt(1));
    Assert.Equal("Gruyère", bi2.Name);
    Assert.Equal(45.5m, bi2.UnitPrice);
    Assert.Equal(1, bi2.Quantity);
```

Assertion Roulette

```
[Fact]
public void UseBasketPipelineOnExpensiveBasket()
{
    // Fixture setup
    var basket = new Basket(
        new BasketItem("Chocolate", 50, 3),
        new BasketItem("Gruyère", 45.5m, 1),
        new BasketItem("Barolo", 250, 2));
    CompositePipe<Basket> pipeline = new BasketPipeline();
    // Exercise system
    var actual = pipeline.Pipe(basket);
    // Verify outcome
    var bi1 = Assert.IsAssignableFrom<BasketItem>(
        actual.ElementAt(0));
    Assert.Equal("Chocolate", bi1.Name);
    Assert.Equal(50, bi1.UnitPrice);
    Assert.Equal(3, bi1.Quantity);

    var bi2 = Assert.IsAssignableFrom<BasketItem>(
        actual.ElementAt(1));
    Assert.Equal("Gruyère", bi2.Name);
    Assert.Equal(45.5m, bi2.UnitPrice);
    Assert.Equal(1, bi2.Quantity);
```

Assertion Roulette

```
[Fact]
public void UseBasketPipelineOnExpensiveBasket()
{
    // Fixture setup
    var basket = new Basket(
        new BasketItem("Chocolate", 50, 3),
        new BasketItem("Gruyère", 45.5m, 1),
        new BasketItem("Barolo", 250, 2));
    CompositePipe<Basket> pipeline = new BasketPipeline();
    // Exercise system
    var actual = pipeline.Pipe(basket);
    // Verify outcome
    var bi1 = Assert.IsAssignableFrom<BasketItem>(
        actual.ElementAt(0));
    Assert.Equal("Chocolate", bi1.Name);
    Assert.Equal(50, bi1.UnitPrice);
    Assert.Equal(3, bi1.Quantity);

    var bi2 = Assert.IsAssignableFrom<BasketItem>(
        actual.ElementAt(1));
    Assert.Equal("Gruyère", bi2.Name);
    Assert.Equal(45.5m, bi2.UnitPrice);
    Assert.Equal(1, bi2.Quantity);
    var bi3 = Assert.IsAssignableFrom<BasketItem>(
        actual.ElementAt(2));
    Assert.Equal("Barolo", bi3.Name);
    Assert.Equal(250, bi3.UnitPrice);
    Assert.Equal(2, bi3.Quantity);

    var d = Assert.IsAssignableFrom<Discount>(
        actual.ElementAt(3));
    Assert.Equal(34.775m, d);

    var v = Assert.IsAssignableFrom<Vat>(
        actual.ElementAt(4));
    Assert.Equal(165.18125m, v);

    var bt = Assert.IsAssignableFrom<BasketTotal>(
        actual.ElementAt(5));
    Assert.Equal(825.90625m, bt);
    // Teardown
}
```

Not DAMP

DAMP assertion using proper identity

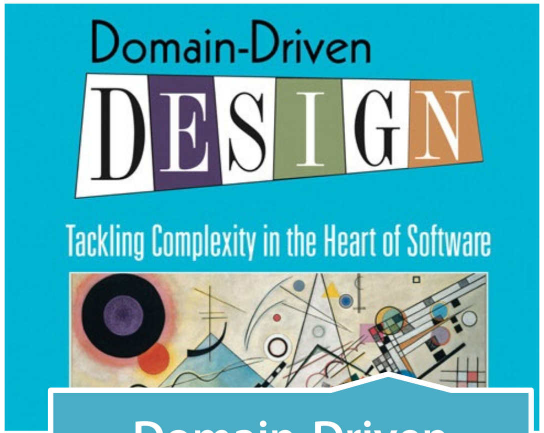
```
[Fact]
public void UseBasketPipelineOnExpensiveBasket()
{
    // Fixture setup
    var basket = new Basket(
        new BasketItem("Chocolate", 50, 3),
        new BasketItem("Gruyère", 45.5m, 1),
        new BasketItem("Barolo", 250, 2));
    CompositePipe<Basket> pipeline = new BasketPipeline();
    // Exercise system
    var actual = pipeline.Pipe(basket);
    // Verify outcome
    var expected = new Basket(
        new BasketItem("Chocolate", 50, 3),
        new BasketItem("Gruyère", 45.5m, 1),
        new BasketItem("Barolo", 250, 2),
        new Discount(34.775m),
        new Vat(165.18125m),
        new BasketTotal(825.90625m));
    Assert.Equal(expected, actual);
    // Teardown
}
```

DAMP

Identity matters

Equals determines identity

Principle of Least Astonishment (POLA)



Domain-Driven Design

Object types

Object type	Identity

Object types

Object type	Identity
Entities	

Object types

Object type	Identity
Entities	Outlasts process lifetime

Object types

Object type	Identity
Entities	Outlasts process lifetime ID

Object types

Object type	Identity
Entities	Outlasts process lifetime ID
Value Objects	

Object types

Object type	Identity
Entities	Outlasts process lifetime ID
Value Objects	Value

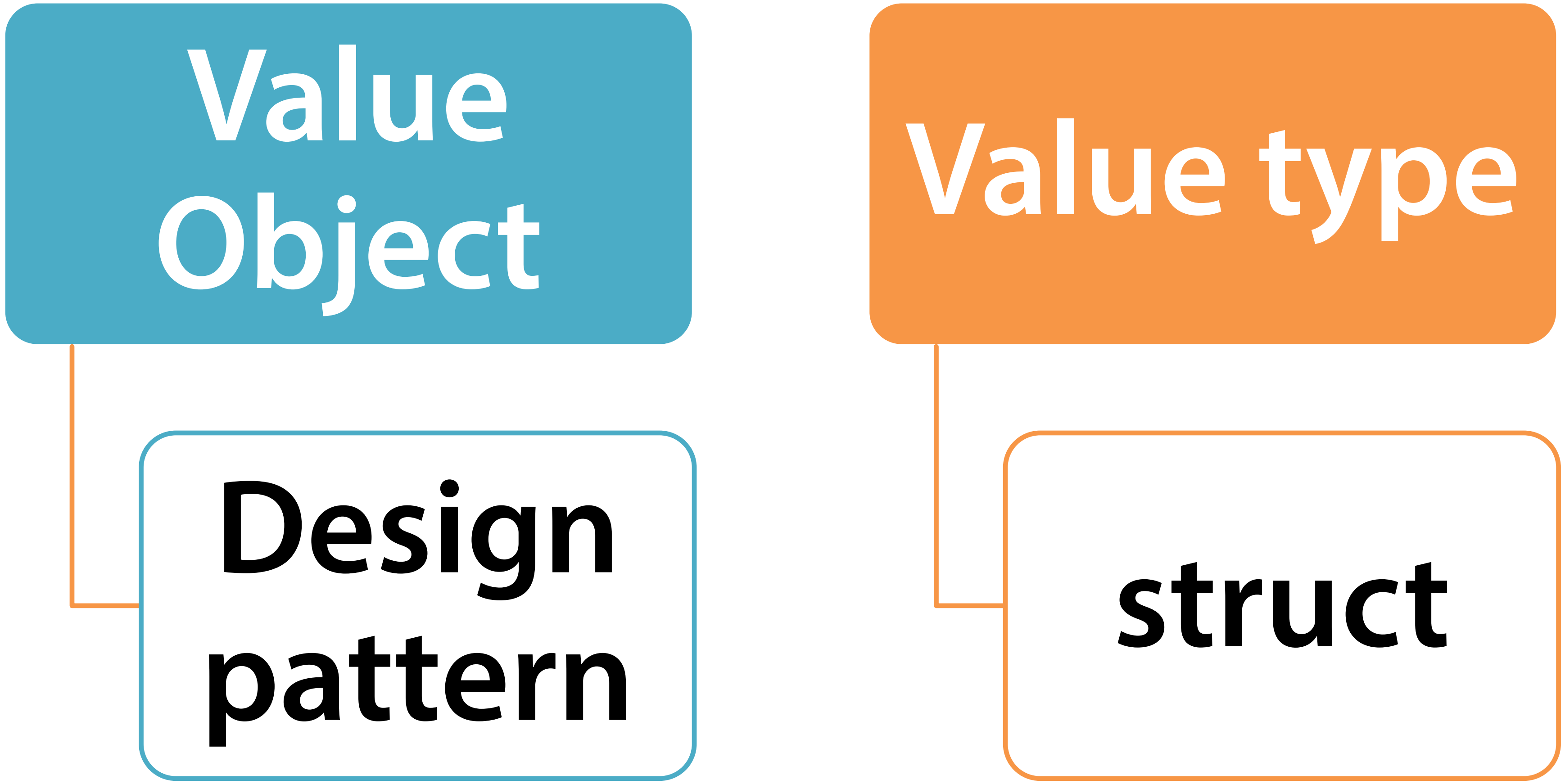
Object types

Object type	Identity
Entities	Outlasts process lifetime ID
Value Objects	Value
Services	

Object types

Object type	Identity
Entities	Outlasts process lifetime ID
Value Objects	Value
Services	Default (reference)

Value Objects



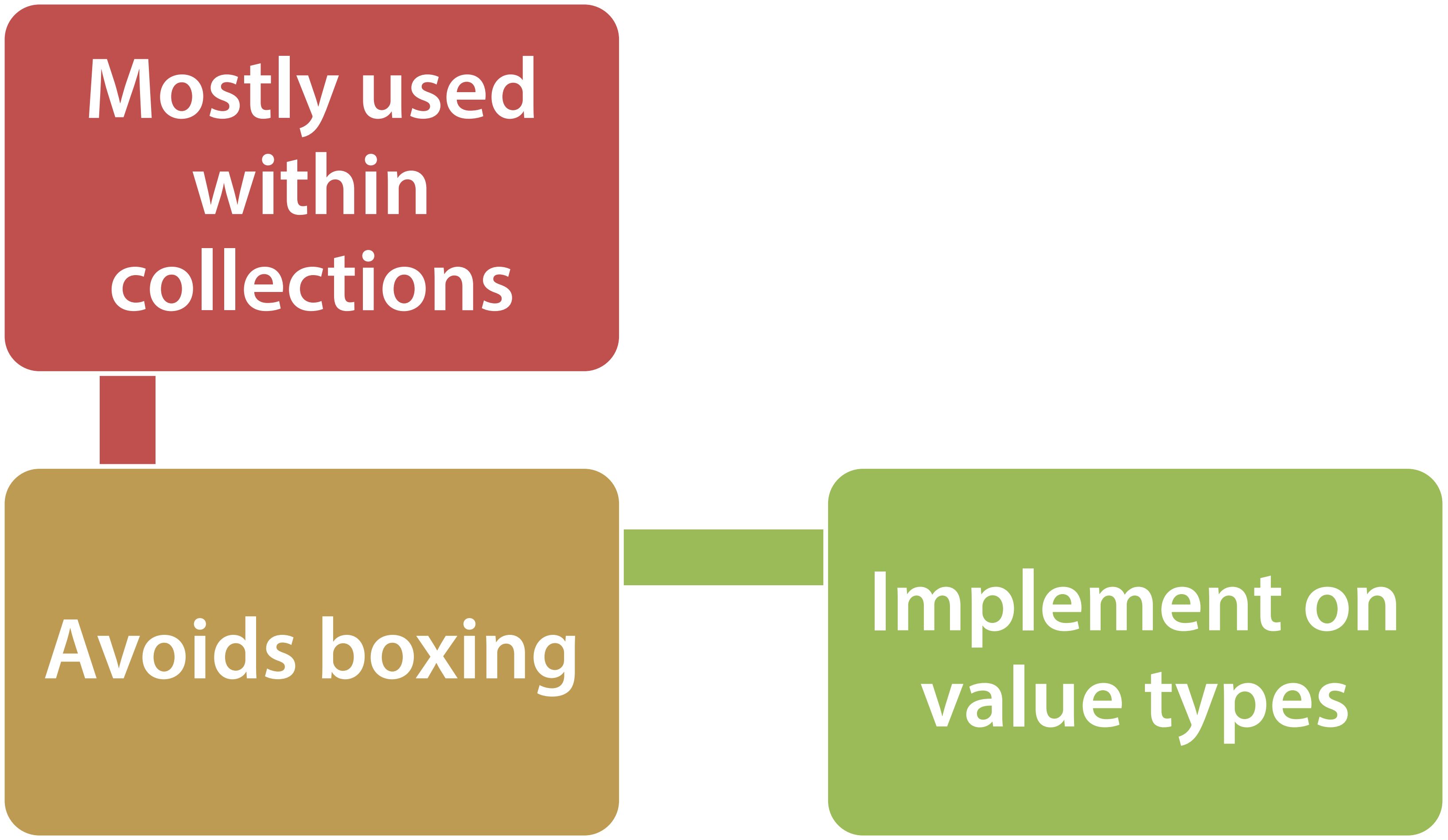
Unit testing Equals on Value Objects

```
[Theory]
[InlineData(1, 1, true)]
[InlineData(2, 1, false)]
[InlineData(2, 2, true)]
public void EqualsOtherTotalReturnsCorrectResult(
    int totalAmount,
    int otherAmount,
    bool expected)
{
    var sut = new BasketTotal(totalAmount);
    var other = new BasketTotal(otherAmount);

    var actual = sut.Equals(other);

    Assert.Equal(expected, actual);
}
```

Unit testing GetHashCode on Value Objects

```
[Theory]
[InlineData(1.1)]
[InlineData(2.5)]
public void GetHashCodeReturnsCorrectResult(
    double total)
{
    var d = (decimal)total;
    var sut = new BasketTotal(d);

    var actual = sut.GetHashCode();

    var expected = d.GetHashCode();
    Assert.Equal(expected, actual);
}
```

IEquatable<T>


Mostly used within collections
Avoids boxing
Implement on value types

Unit testing IEquatable<T>

```
[Theory]
[InlineData(1, 1, true)]
[InlineData(2, 1, false)]
[InlineData(2, 2, true)]
public void EqualsOtherTotalReturnsCorrectResult(
    int totalAmount,
    int otherAmount,
    bool expected)
{
    var sut = new BasketTotal(totalAmount);
    var other = new BasketTotal(otherAmount);

    var actual = sut.BothEquals(other);

    Assert.True(actual.All(expected.Equals));
}
```

BothEquals

```
public static IEnumerable<bool> BothEquals<T>(
    this T sut, T other)
    where T : IEquatable<T>
{
    yield return sut.Equals((object)other);
    yield return sut.Equals(other);
}
```

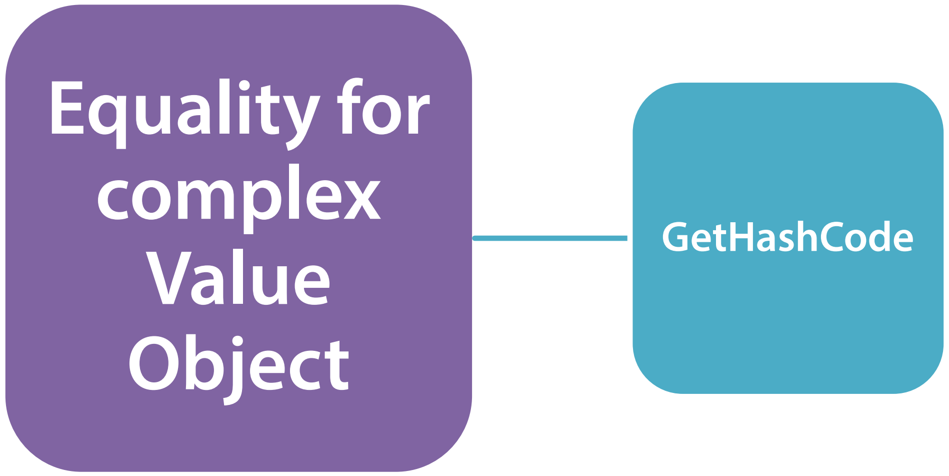
Structural Inspection without properties

```
[Theory]
[InlineData(1.1)]
[InlineData(2.3)]
public void SutCorrectlyConvertsToDecimal(
    double d)
{
    var expected = (decimal)d;
    var sut = new BasketTotal(expected);

    decimal actual = sut;

    Assert.Equal(expected, actual);
}
```

Converting to decimal

```
public static implicit operator decimal(
    BasketTotal basketTotal)
{
    return basketTotal.total;
}
```

Demo



Demo recap

All values must match

More values require more test cases

but not more test methods

GetHashCode

Unit testing Equals on Entities

```
[Theory]
[InlineData(1, 1, true)]
[InlineData(2, 1, false)]
[InlineData(2, 2, true)]
public void EqualsOtherUserReturnsCorrectResult(
    int sutId,
    int otherId,
    bool expected)
{
    var sut = new User(sutId, "Dummy name");
    var other = new User(otherId, "Dummy name");

    var actual = sut.Equals(other);

    Assert.Equal(expected, actual);
}
```

Unit testing Equals on Entities

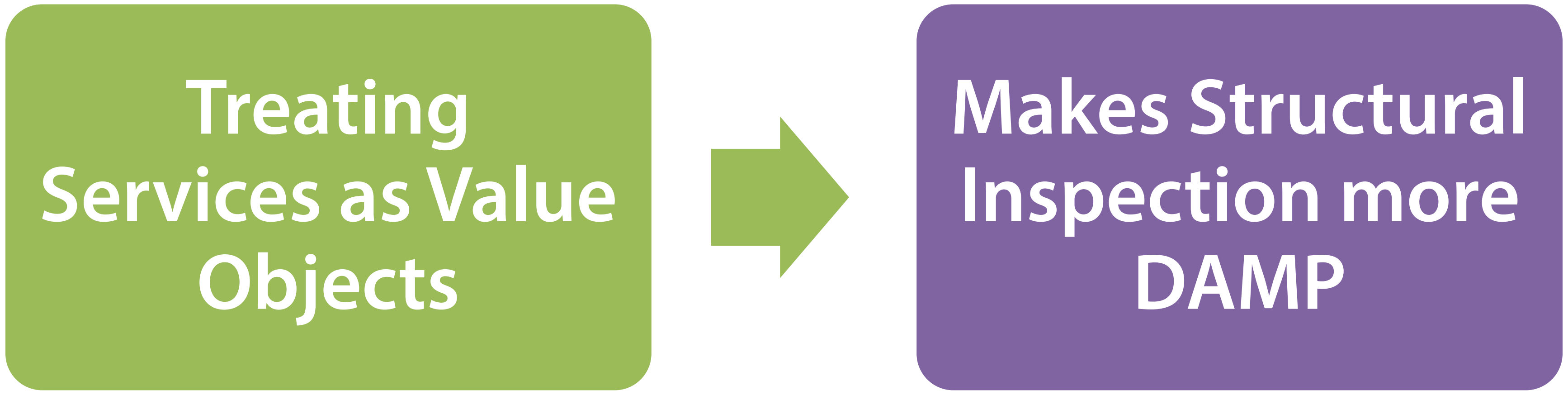
```
[Theory]
[InlineData(1, 1, true)]
[InlineData(2, 1, false)]
[InlineData(2, 2, true)]
public void EqualsOtherUserReturnsCorrectResult(
    int sutId,
    int otherId,
    bool expected)
{
    var sut = new User(sutId, "Dummy name");
    var other = new User(otherId, "Other dummy name");

    var actual = sut.Equals(other);

    Assert.Equal(expected, actual);
}
```

Unit testing Equals on Services

This page intentionally left blank

Demo

Demo recap


Treating Services as Value Objects
Makes Structural Inspection more DAMP

Summary

- Value Objects
  - Override Equals
  - Unit test Equals
- Entities
- Services
  - Value Object identity?