DOTNEXT

Pragmatic Unit Testing

Vladimir Khorikov

http://enterprisecraftsmanship.com

@vkhorikov

Cargo cult unit testing

Vladimir Khorikov

http://enterprisecraftsmanship.com

Author at Pluralsight

Goals of Testing

~~Unit testing = Better design~~

Sustainable growth of software project

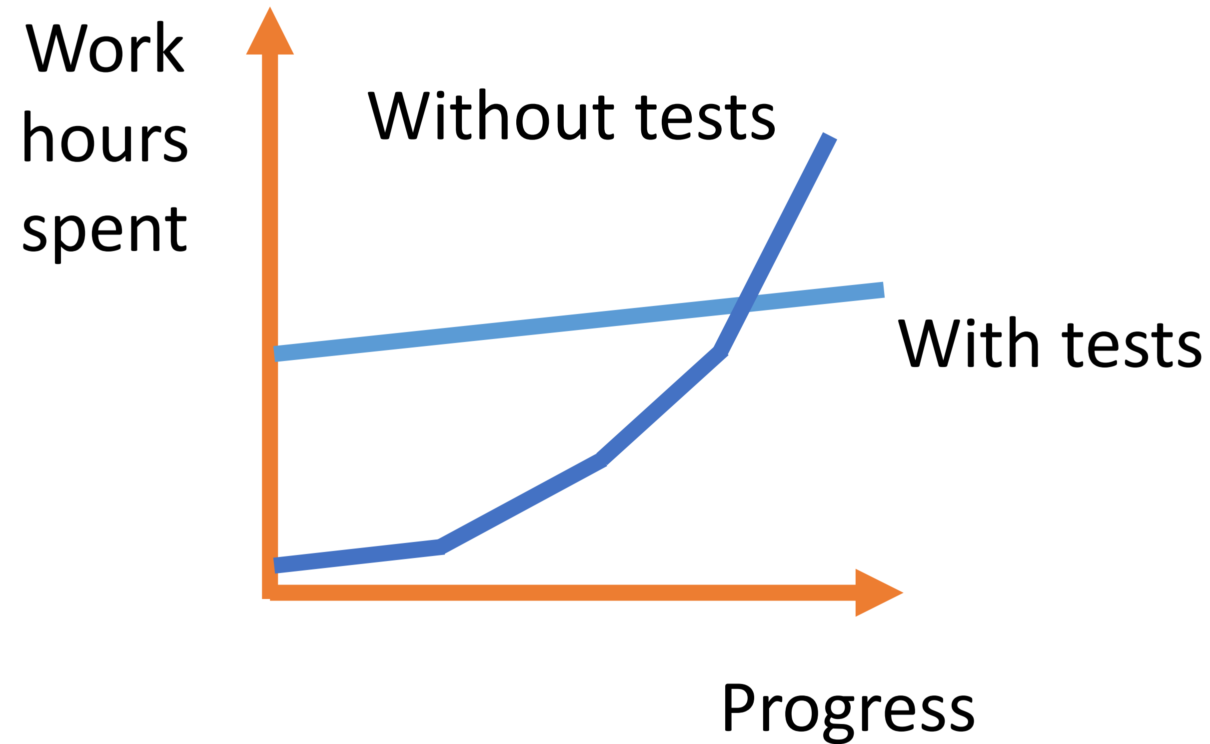
Safety net

Changes don’t break existing functionality

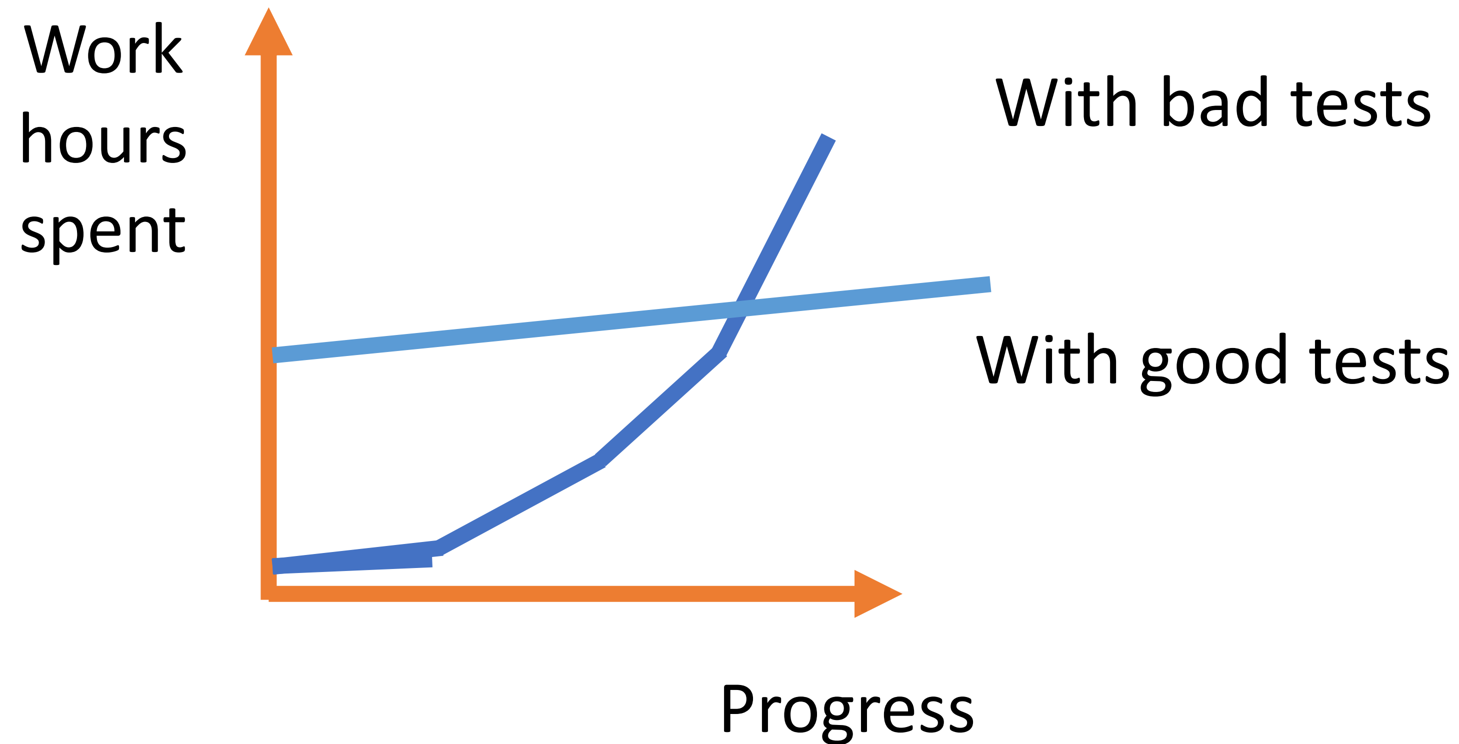
Move with a faster pace

Maintain low amount of technical debt

Focus on the long term



Focus on the long term

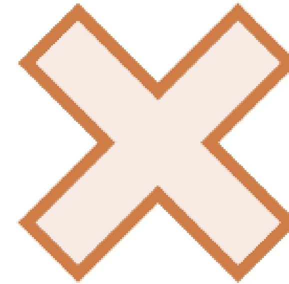


Well, that’s life

You can sustain the development speed!

All Tests are Not Created Equal

Contribute to the software quality and the safety net



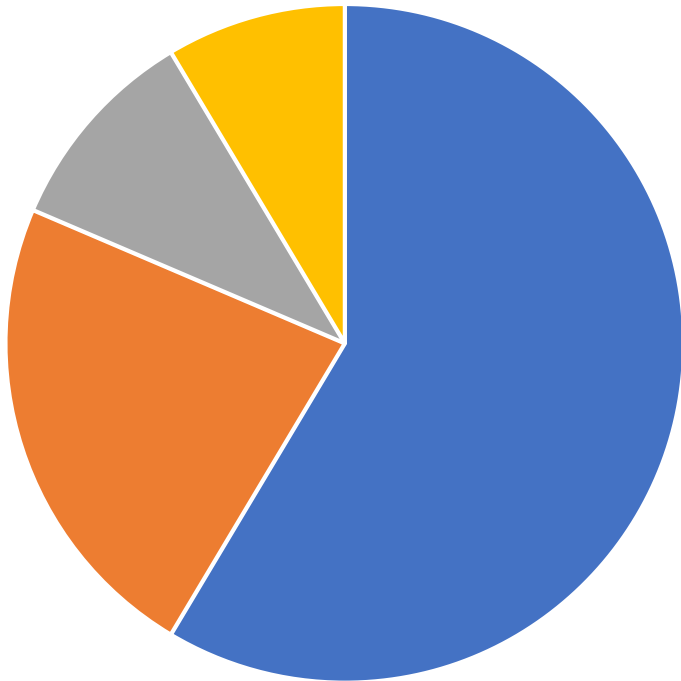
Raise false alarms

Don’t catch regressions

Provide slow feedback

How to evaluate your test suite?

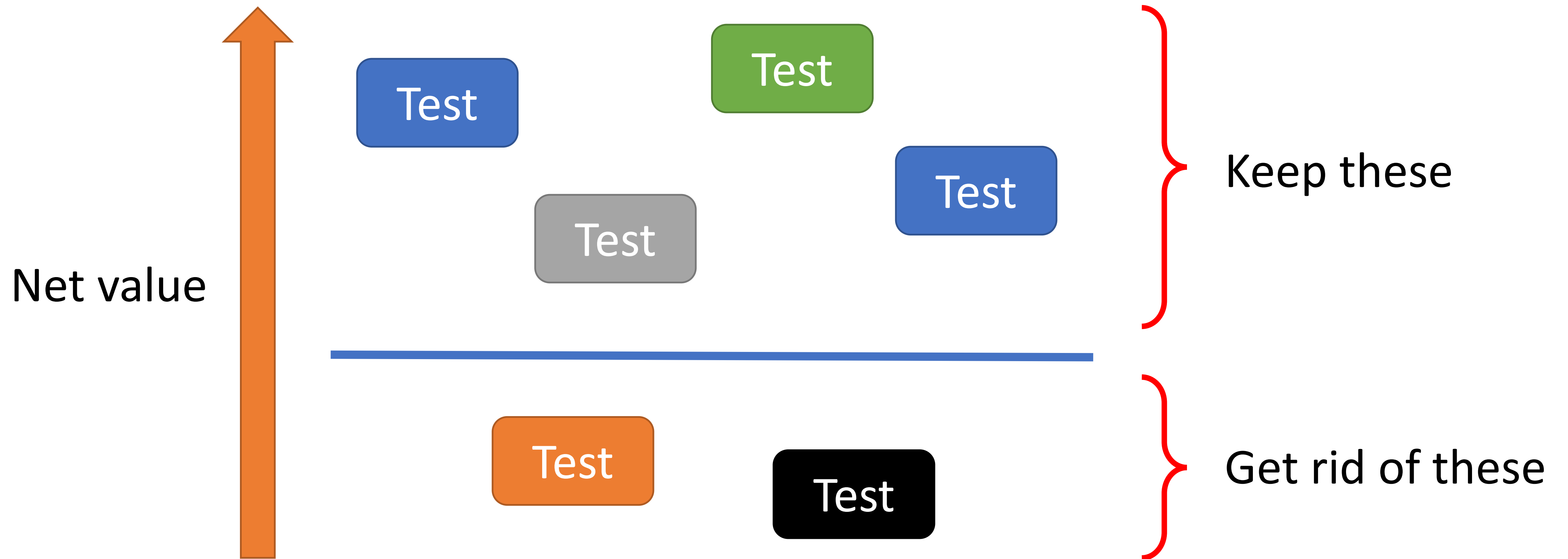
Coverage Metrics



Good negative indicator

Bad positive indicator

Pragmatic Approach to Unit Testing



What Makes a Test Valuable?

Protection against regressions

Resistance to refactoring

Fast feedback

Maintainability

Protection Against Regressions

The more code is exercised, the better the protection

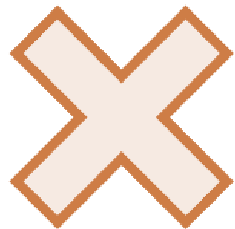
The more important the code, the better

External libraries and systems count too

Testing trivial code is not worth it

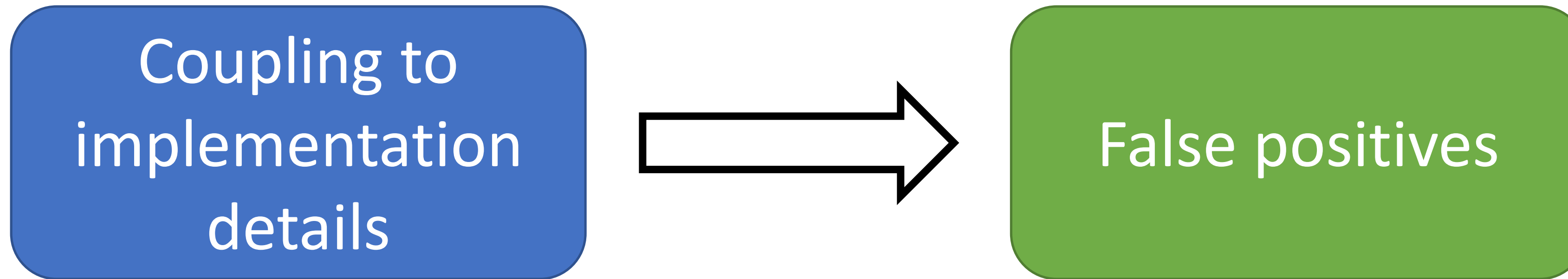
Resistance to refactoring

False positive = False alarm



Dilute the ability to spot a problem

Resistance to refactoring



Decouple tests from implementation details as much as possible

What Makes a Test Valuable?

Protection against regressions

Resistance to refactoring

What Makes a Test Valuable?

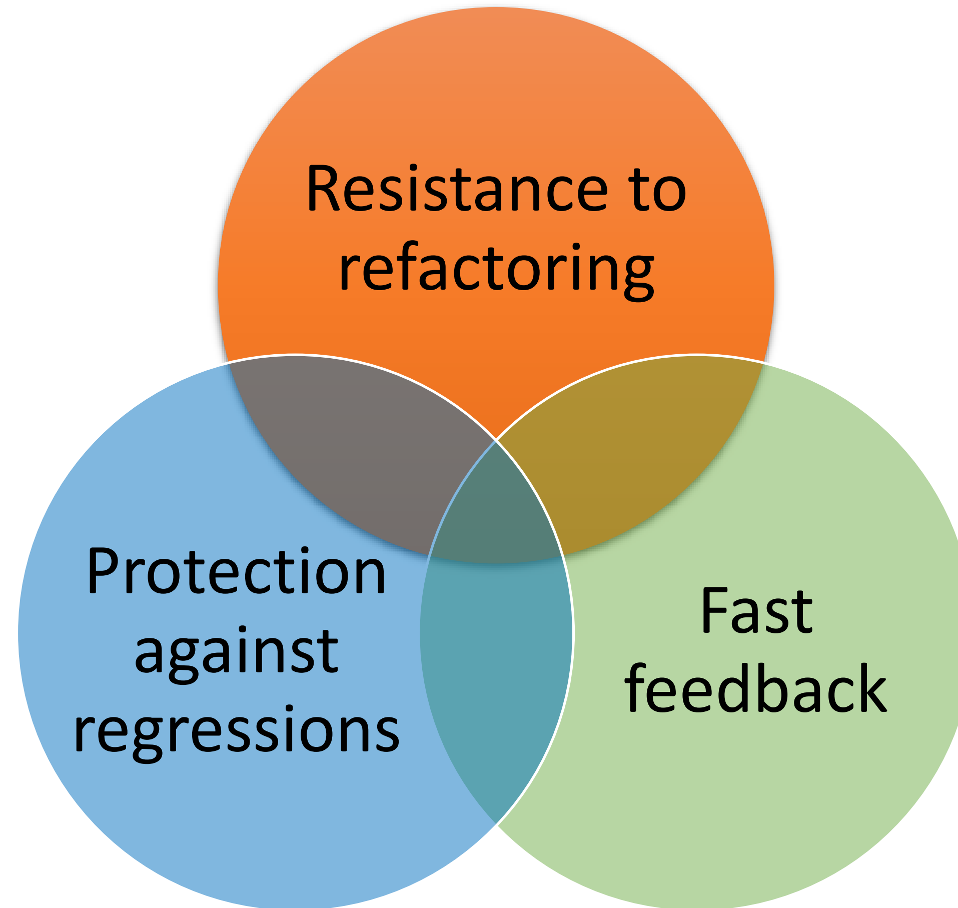
Table of error types		Functionality is	
		Correct	Broken
Test result	Test passes	Correct inference (True Negatives)	Type II error (False Negative)
	Test fails	Type I error (False Positive)	Correct inference (True Positives)

What Makes a Test Valuable?

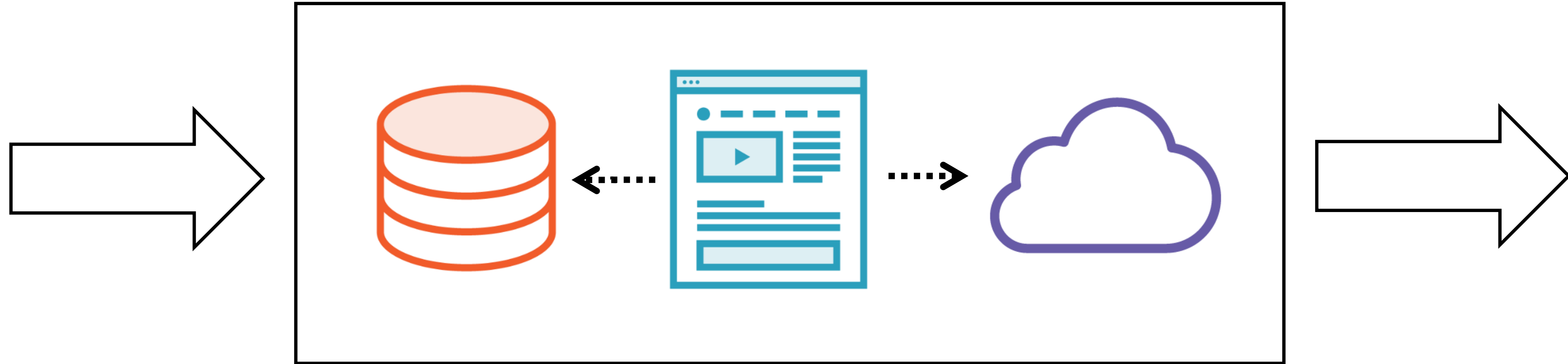
Protection against regressions	Resistance to refactoring
Fast feedback	Maintainability

0..1 * 0..1 * 0..1 * 0..1 = Value estimate

What Makes a Valuable Test: Examples



End-to-end Tests

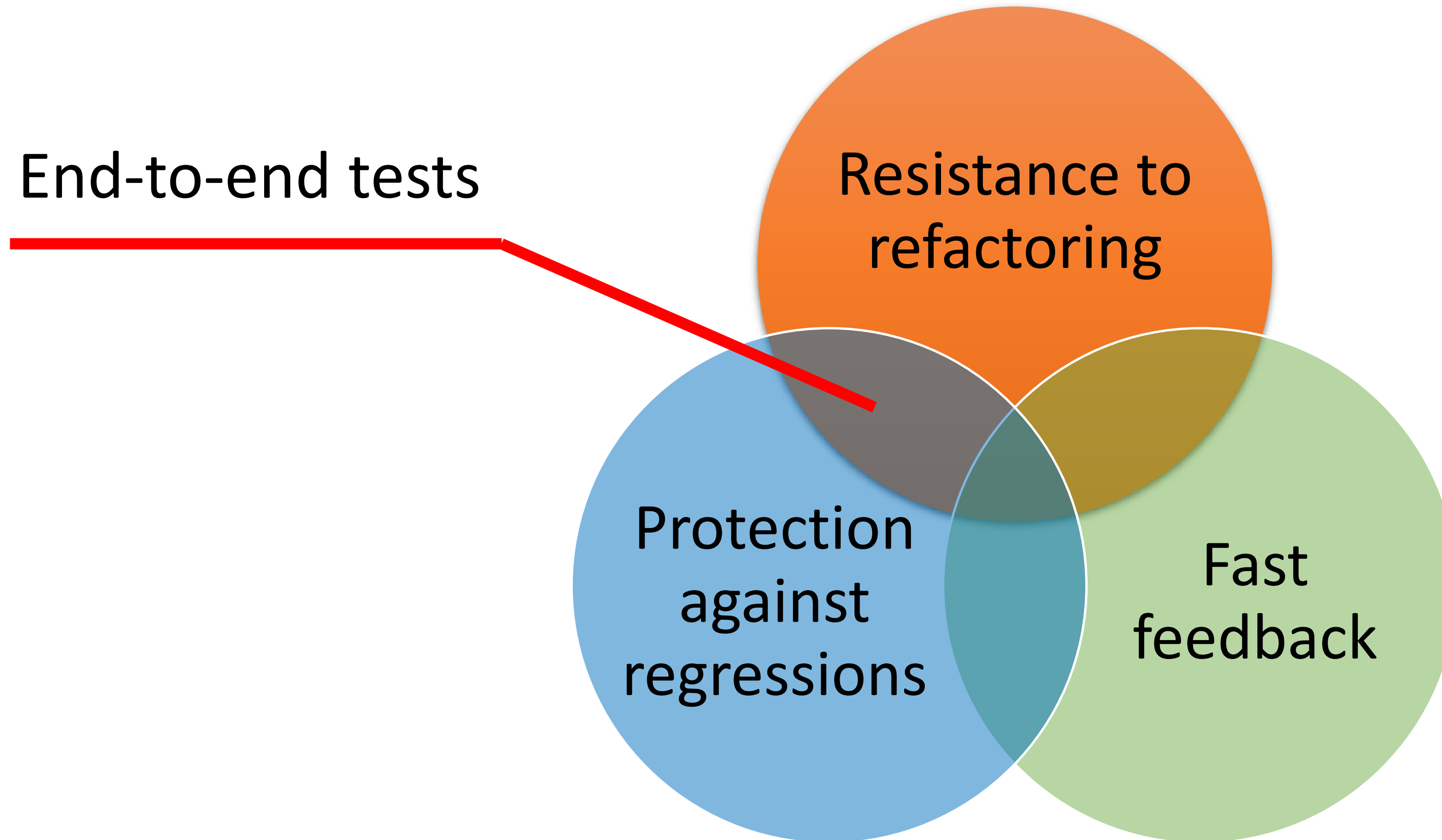


Best protection against regressions

Immune to false positives

Slow feedback

What Makes a Valuable Test: Examples



Trivial Test

```
public class User
{
    public string Name { get; set; }
}
```

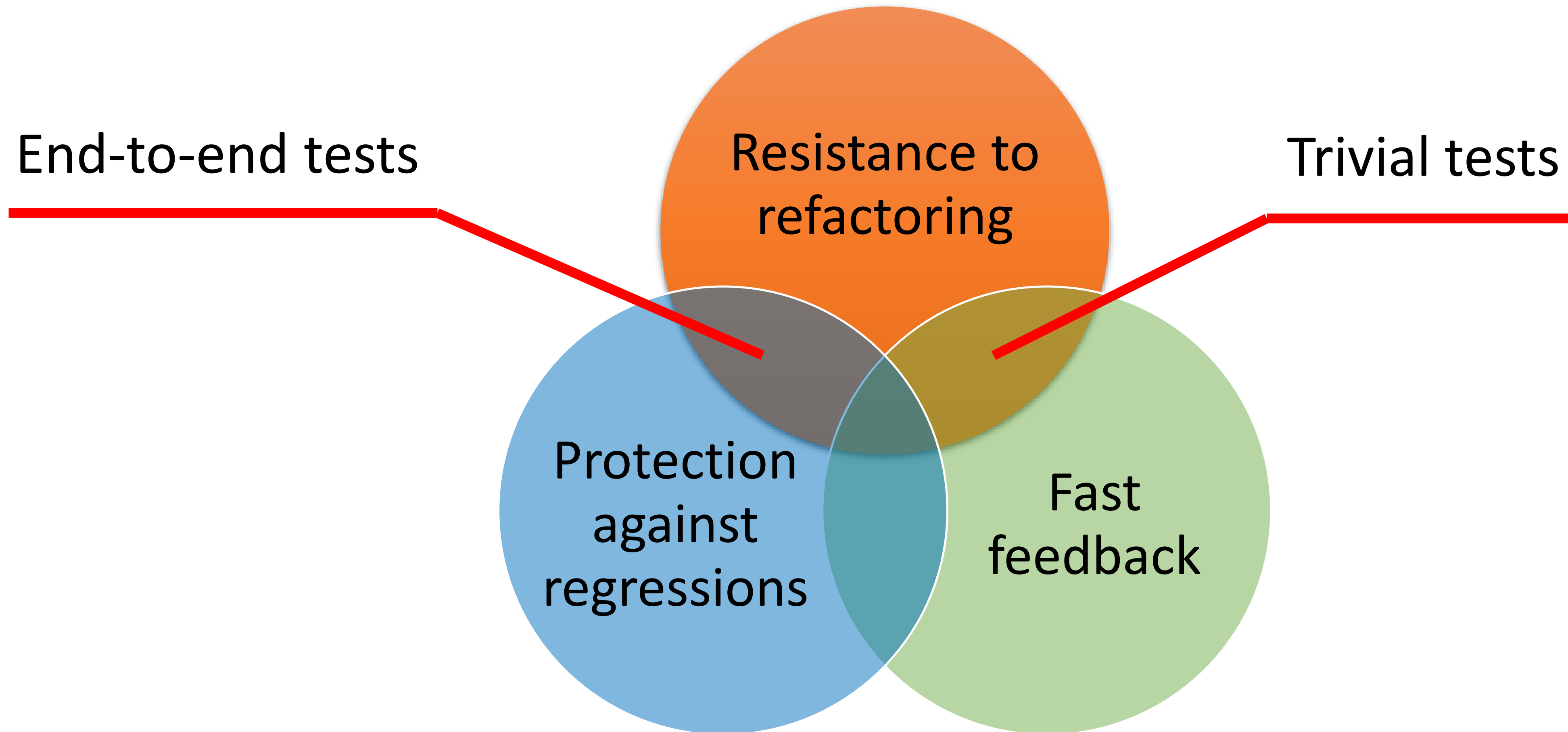
```
[Fact]
public void Test()
{
    var user = new User();

    user.Name = "John Smith";

    Assert.Equal("John Smith", user.Name);
}
```

Fast feedback

Good resistance to refactoring

Unlikely to catch a regression error

What Makes a Valuable Test: Examples



Brittle Tests

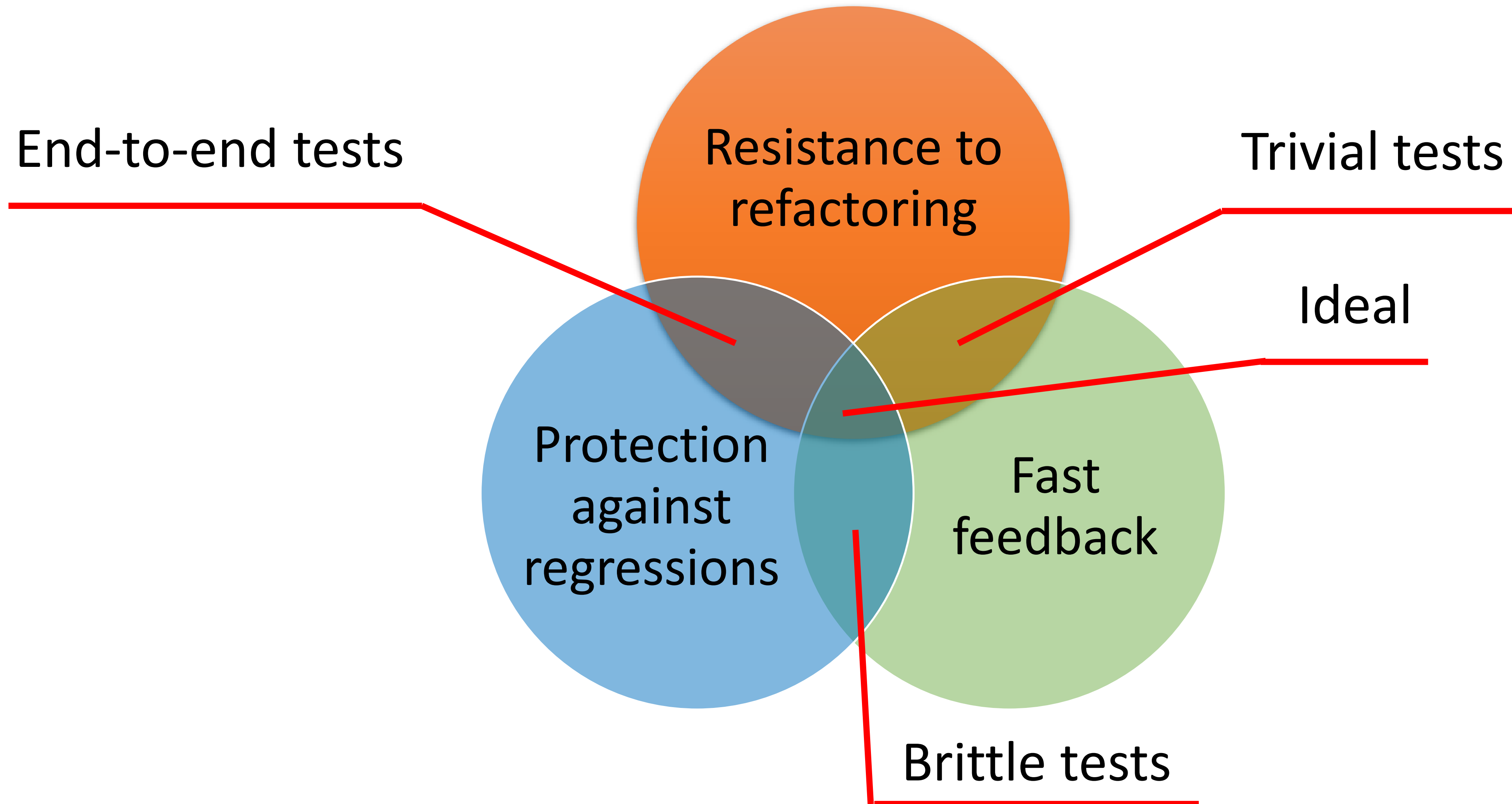
```
public class UserRepository
{
    public User GetById(int id)
    {
        /* … */
    }

    public string LastExecutedSqlStatement
    { get; private set; }
}
```

```
[Fact]
public void GetById_executes_correct_SQL_code()
{
    var repository = new UserRepository();

    User = repository.GetById(5);

    Assert.Equal(
        "SELECT * FROM dbo.[User] WHERE UserID = 5",
        repository.LastExecutedSqlStatement);
}
```

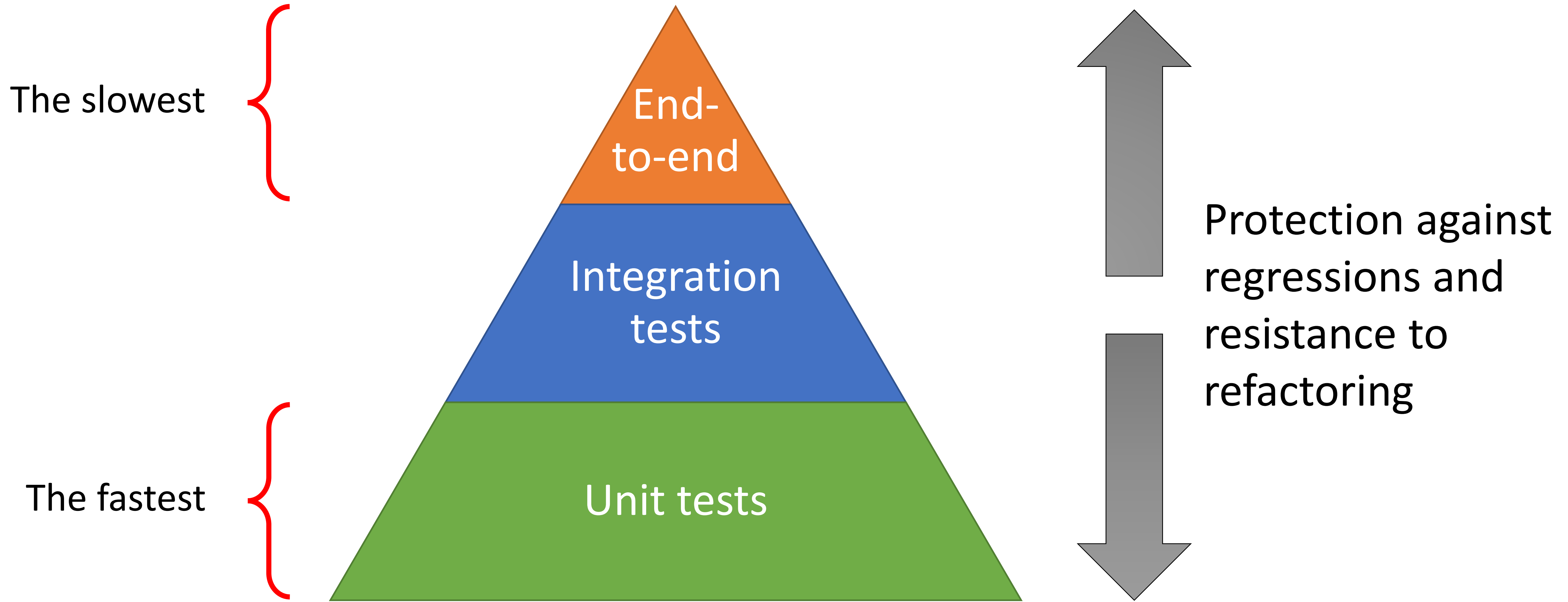
```
SELECT * FROM dbo.[User] WHERE UserID = 5
SELECT * FROM dbo.User WHERE UserID = 5
SELECT UserID, Name, Email FROM dbo.[User] WHERE UserID = 5
SELECT * FROM dbo.[User] WHERE UserID = @UserID
```

Coupling to implementation details

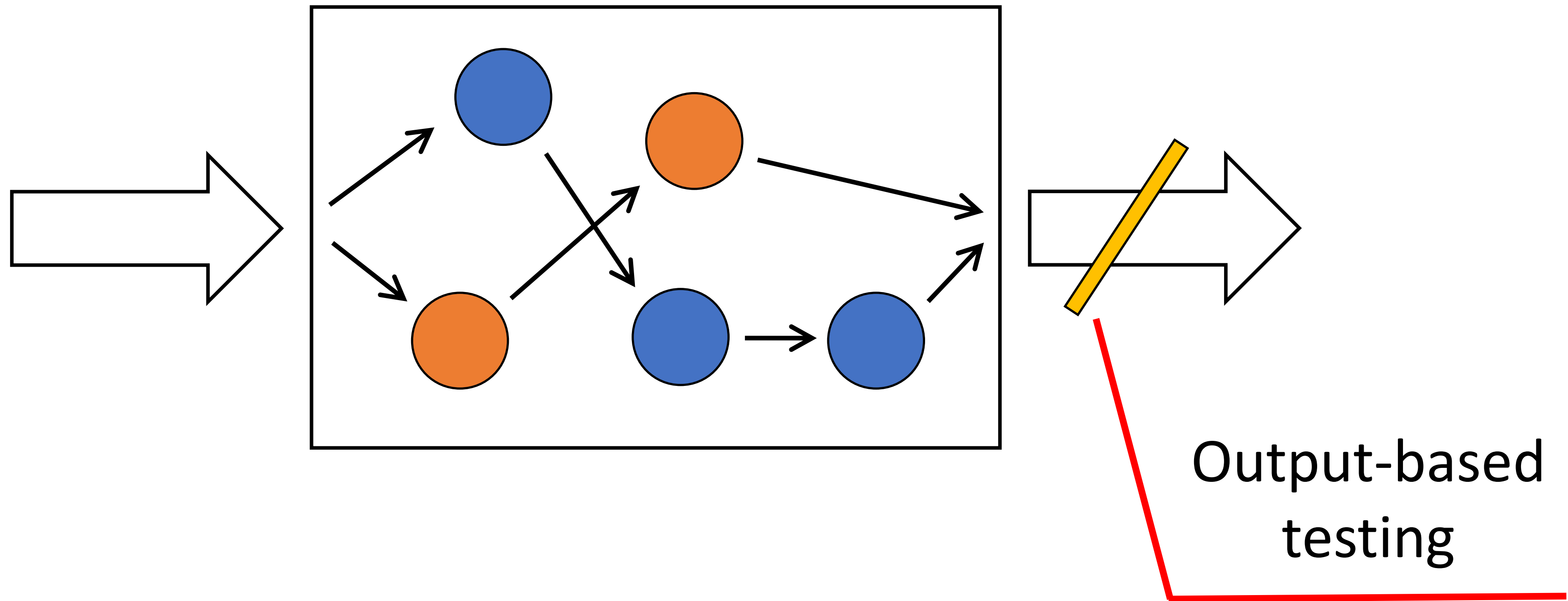
What Makes a Valuable Test: Examples



Test Pyramid



Types of Testing



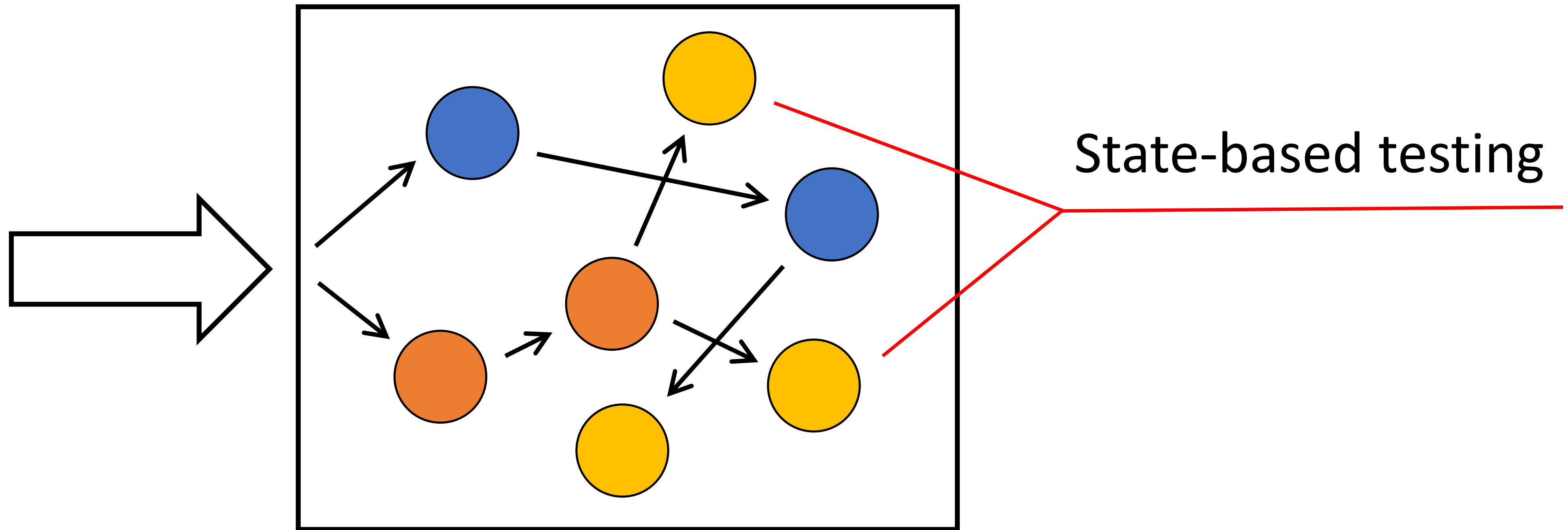
Output-based Testing

```
public class PriceEngine
{
    public decimal CalculateDiscount(
        params Product[] product)
    {
        decimal discount = product.Length * 0.01m;
        return Math.Min(discount, 0.2m);
    }
}
```

```
[Fact]
public void Test()
{
    Product product1 = new Product("Hand wash");
    Product product2 = new Product("Shampoo");
    var engine = new PriceEngine();

    decimal discount = engine.CalculateDiscount(
        product1, product2);

    Assert.Equal(0.02m, discount);
}
```

Functional style

Types of Testing

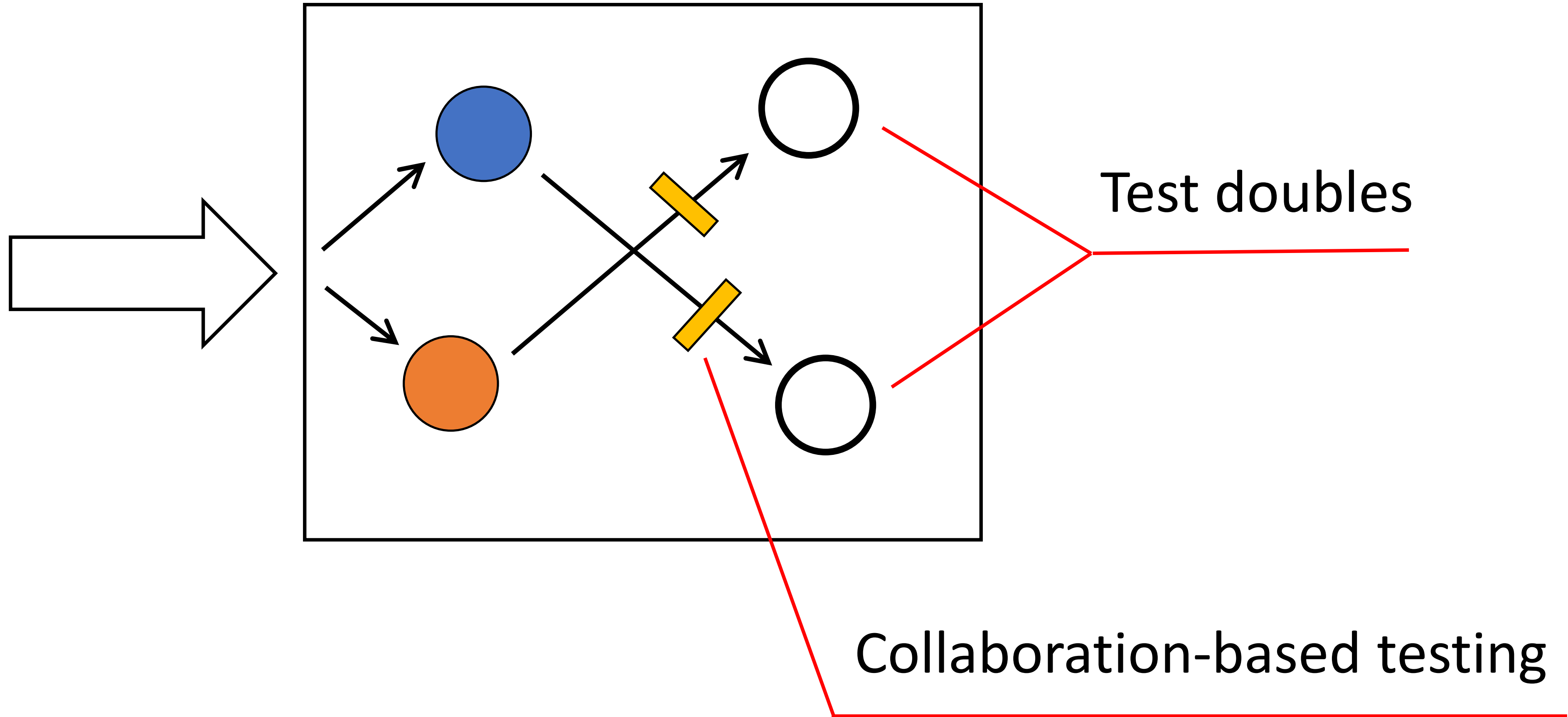


State-based Testing

```csharp
public class Order
{
    private readonly List<Product> _products;
    public IReadOnlyList<Product> Products
        => _products.ToList();

    public void AddProduct(Product product)
    {
        _products.Add(product);
    }
}
```

```csharp
[Fact]
public void Test()
{
    Product product = new Product("Hand wash");
    Order order = new Order();

    order.AddProduct(product);

    Assert.Equal(1, order.Products.Count);
    Assert.Equal(product, order.Products[0]);
}
```

Types of Testing



Collaboration-based Testing

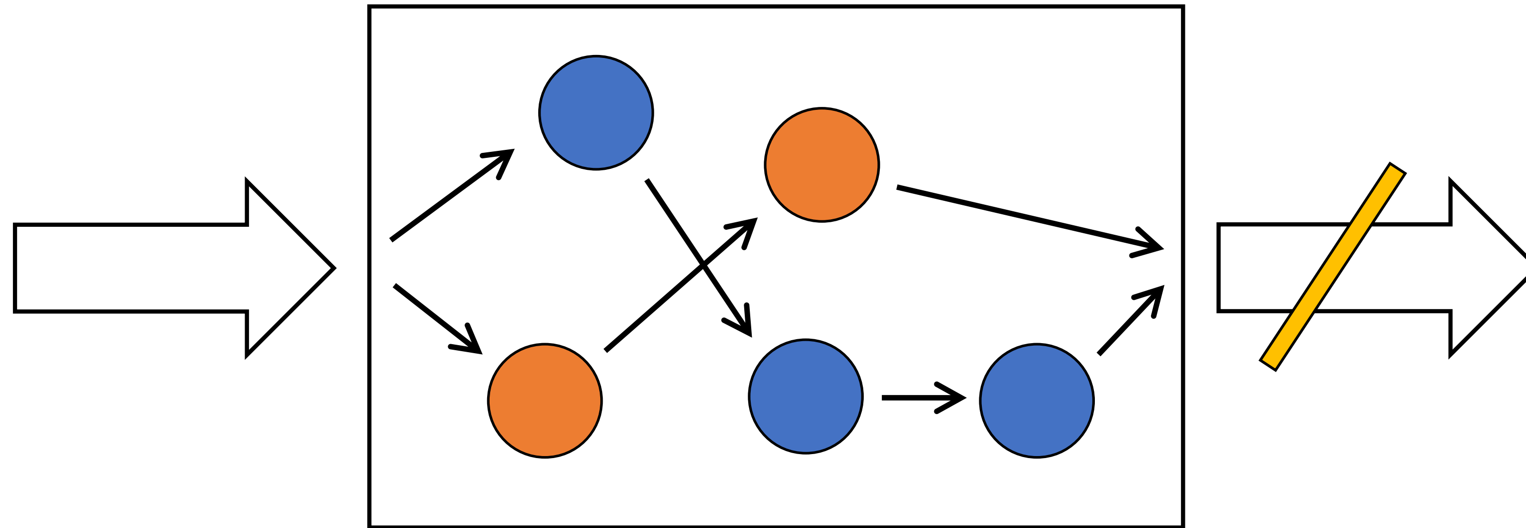
```
public class OrderService
{
    public void Submit(Order order,
        IDatabase database)
    {
        database.Save(order);
    }
}
```

```
[Fact]
public void Test()
{
    var order = new Order();
    var service = new OrderService();
    var mock = new Mock<IDatabase>();

    service.Submit(order, mock.Object);

    mock.Verify(x => x.Save(order));
}
```

Types of Testing: Comparison

Types of testing	Valuable test	
Output-based testing	Protection against regressions	
State-based testing	Resistance to refactoring	
Collaboration-based testing	Fast feedback	
	Maintainability	

Output-based Testing

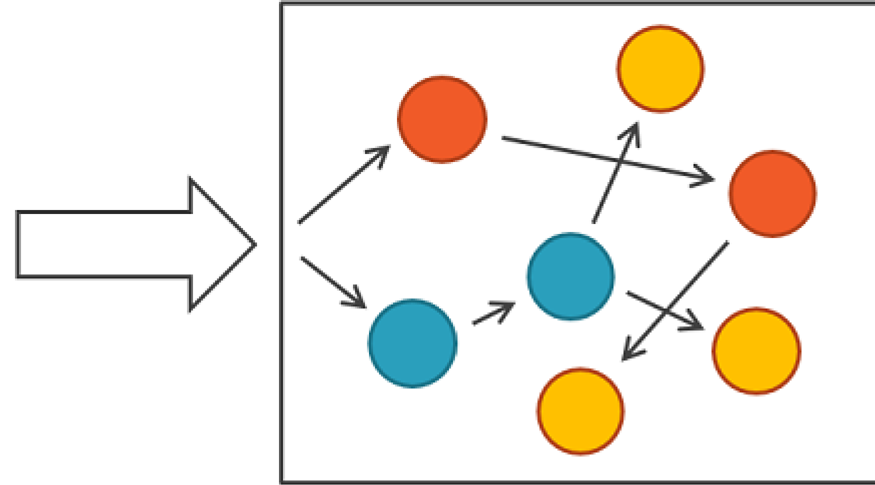


Best protection against false positives

Easy to maintain

Only suitable for functional code

State-based Testing

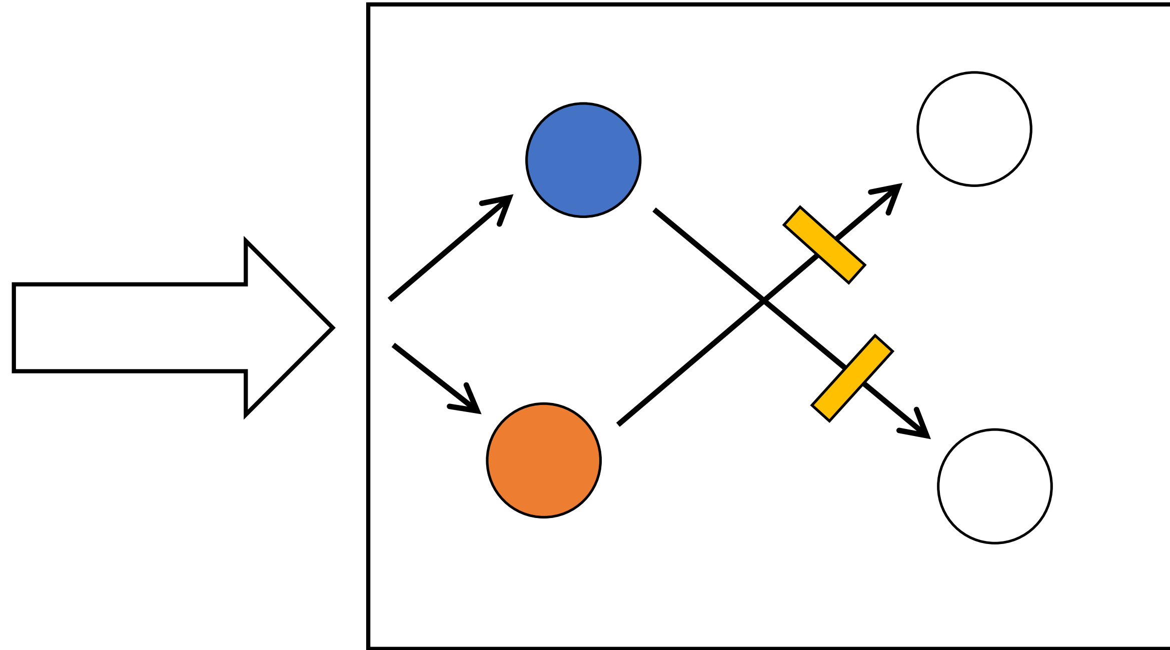


Good protection against false positives

Should verify through the public API

Reasonable maintenance cost

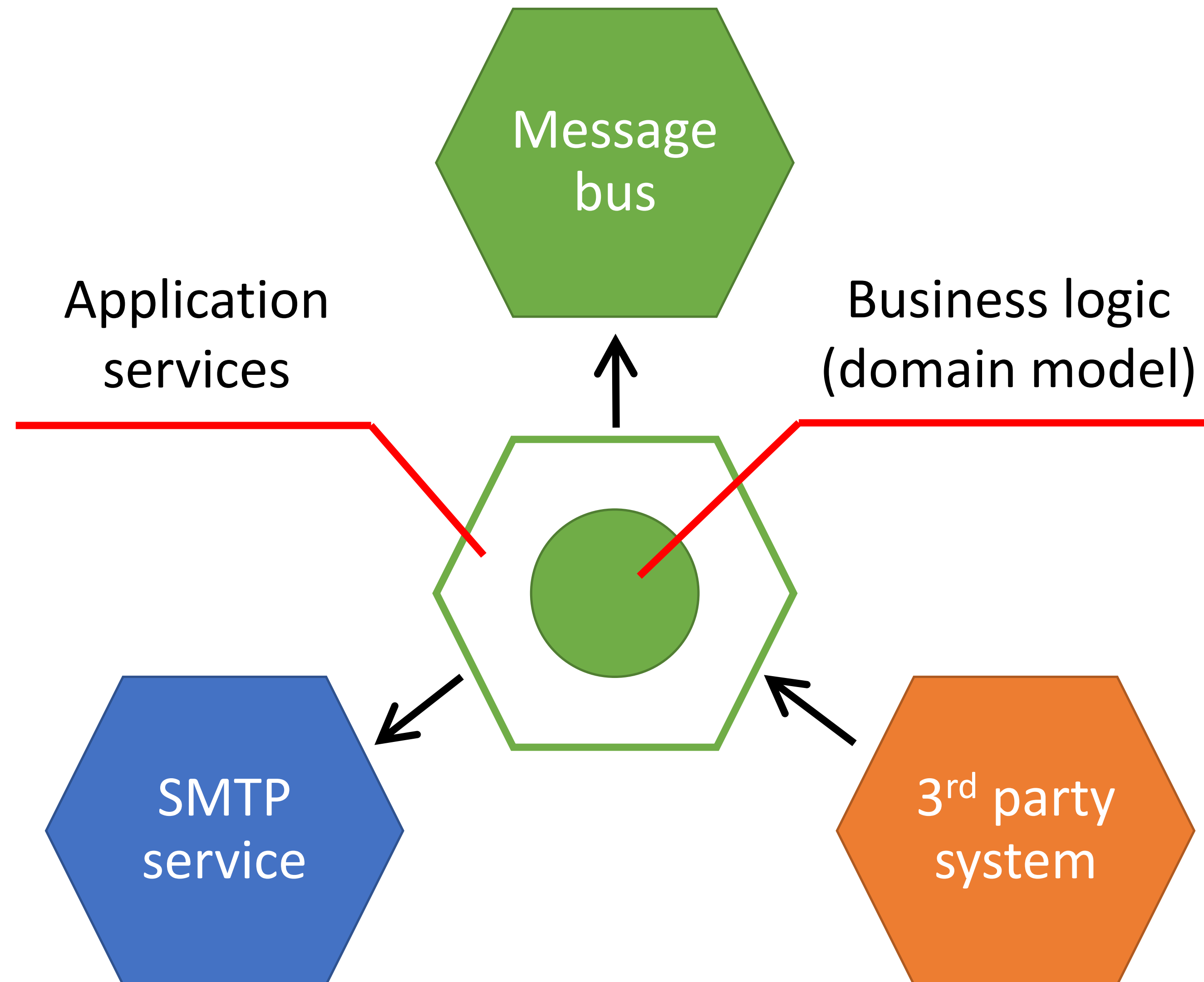
Collaboration-based Testing



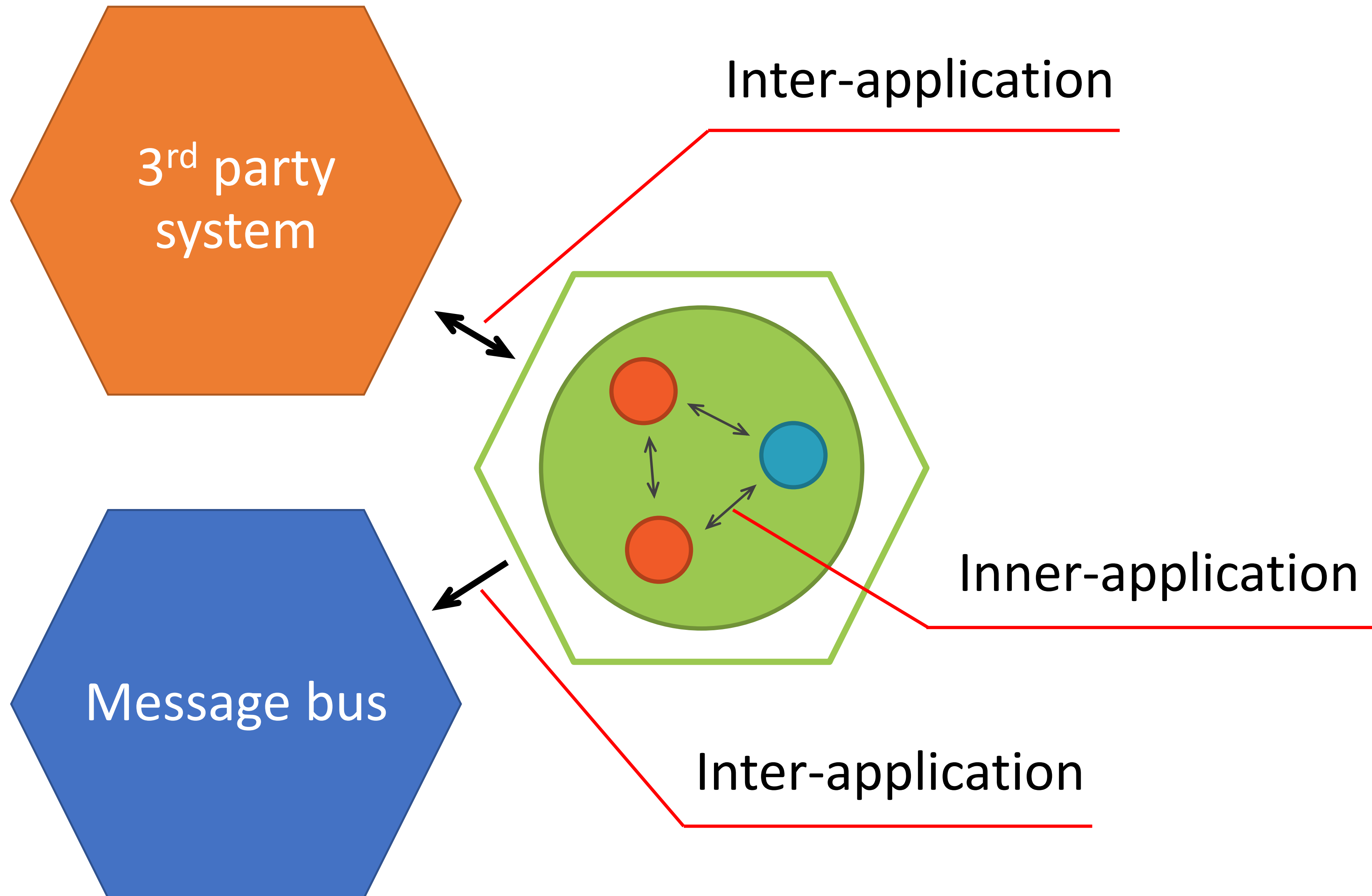
Maintainability is worse

Resistance to refactoring can be much worse

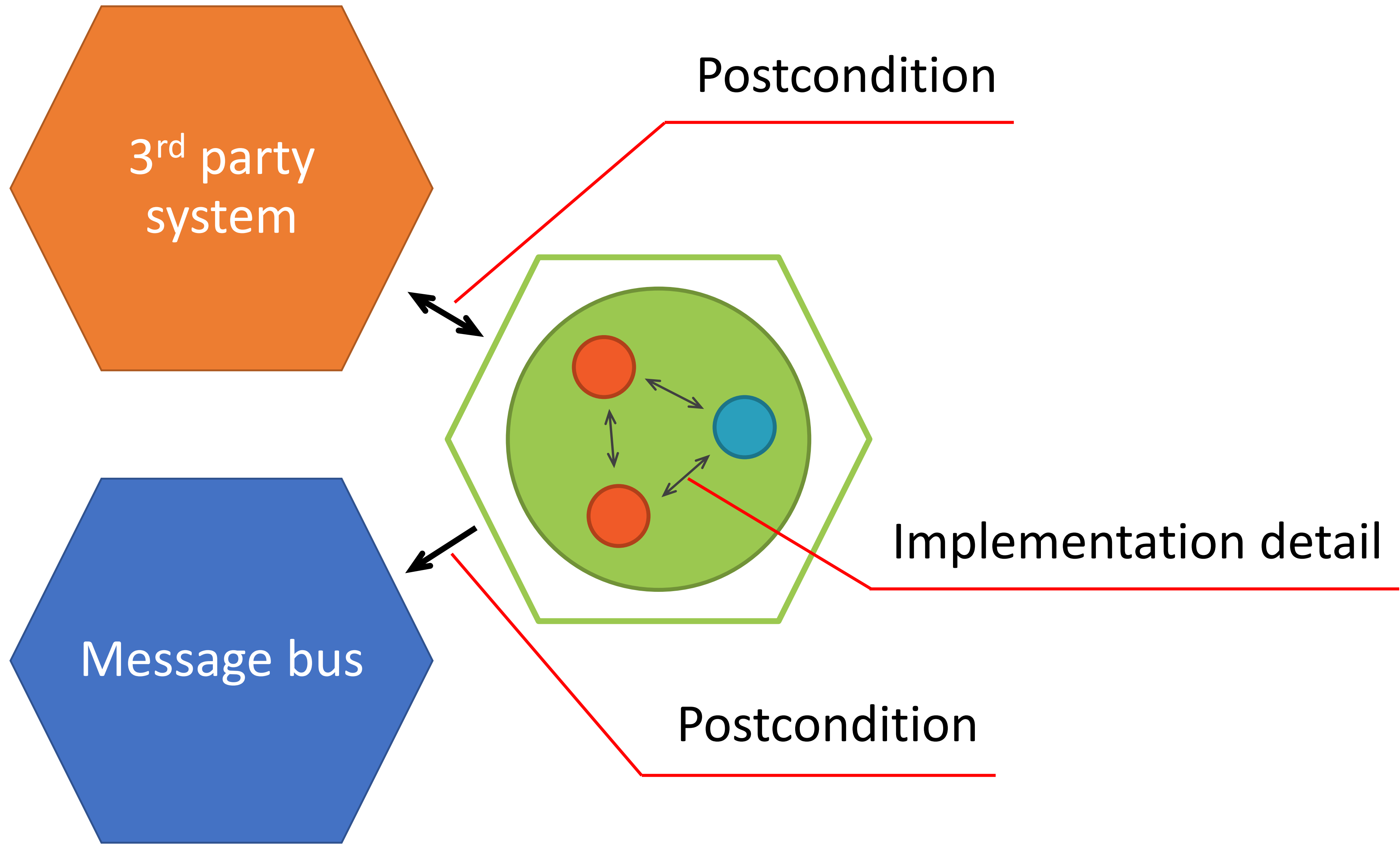
Implementation Detail vs Observable Behavior



Collaboration-based Testing



Collaboration-based Testing



Collaboration-based Testing

```
public class Order {
    private readonly IUser _user;

    public Order(IUser user) {
        _user = user;
    }

    public void AddProduct(Product product) {
        _products.Add(product);
        _user.UpdateLastBoughtProduct(product);
    }
}
```

```
[Fact]
public void Test()
{
    var mock = new Mock<IUser>();
    var order = new Order(mock.Object);
    var product = new Product("M0359");

    order.AddProduct(product);

    mock.Verify(x => x.UpdateLastBoughtProduct(product));
}
```

Collaboration inside the application

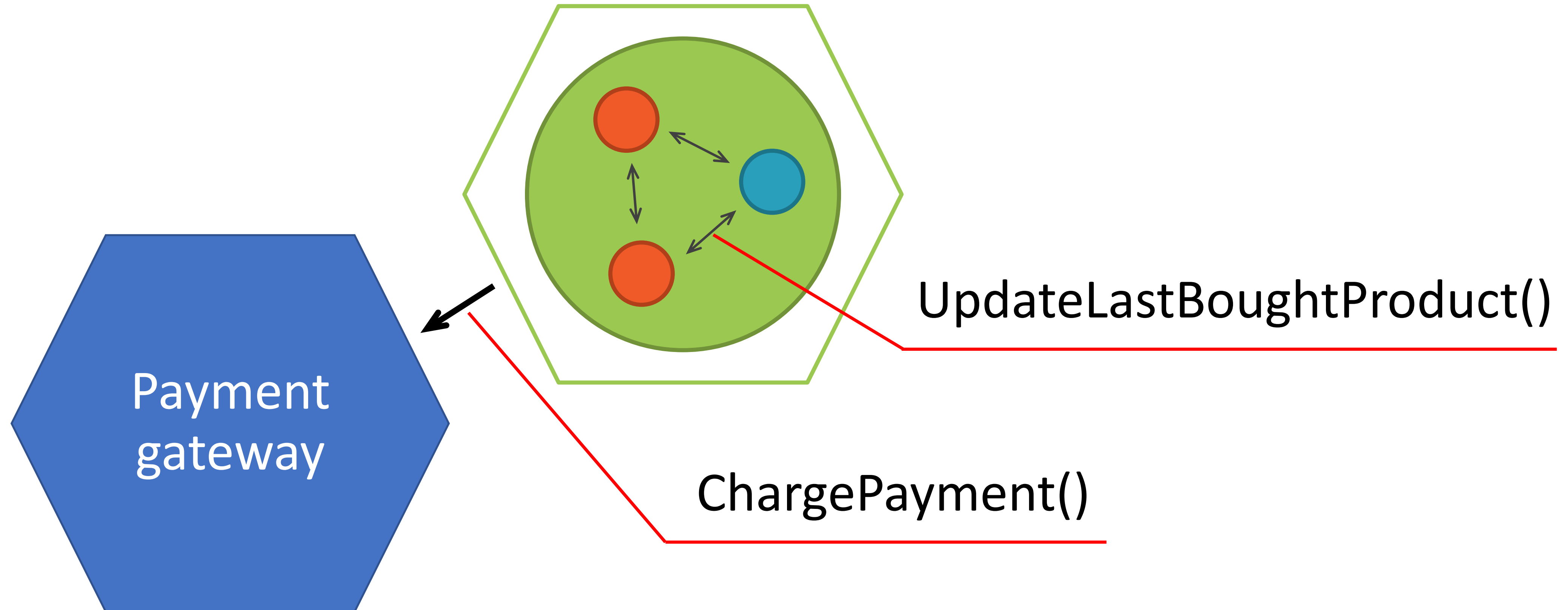
Collaboration-based Testing

```csharp
public class OrderService {
    private readonly IPaymentGateway _gateway;

    public OrderService(IPaymentGateway gateway) {
        _gateway = gateway;
    }

    public void Submit(Order order) {
        _gateway.ChargePayment(order.TotalAmount);
    }
}
```

```csharp
[Fact]
public void Test()
{
    var mock = new Mock<IPaymentGateway>();
    var order = new Order(100);
    var service = new OrderService(mock.Object);

    service.Submit(order);

    mock.Verify(x => x.ChargePayment(100m));
}
```

Collaboration between applications

Collaboration-based Testing



Types of Testing: Comparison

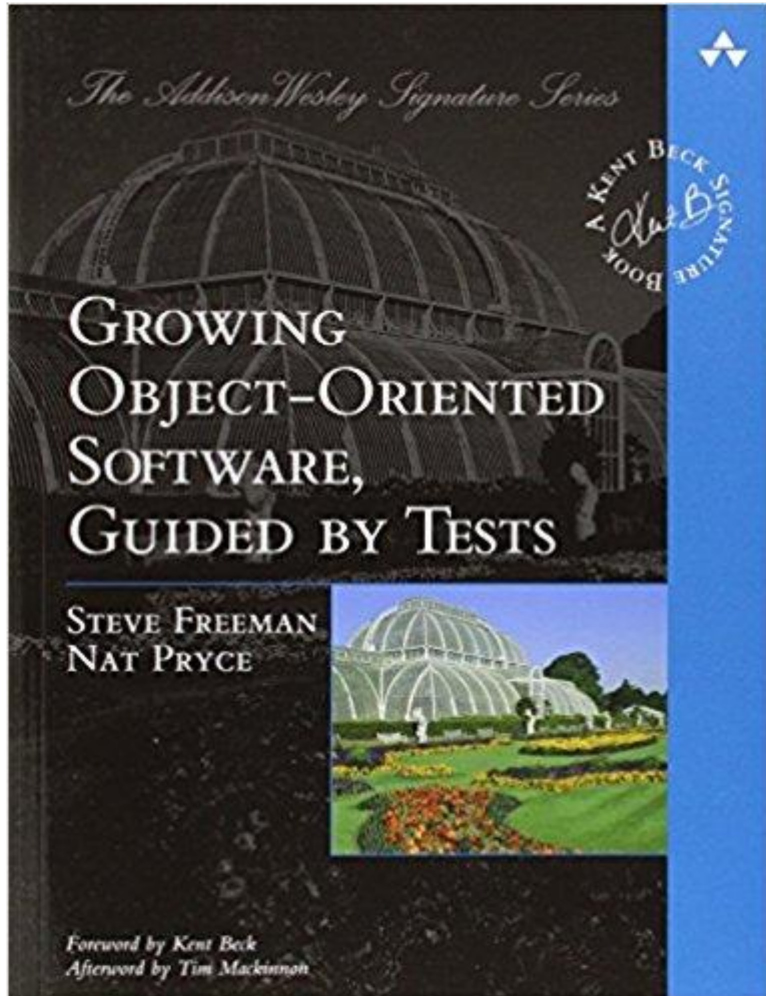
Types of testing

Output-based testing

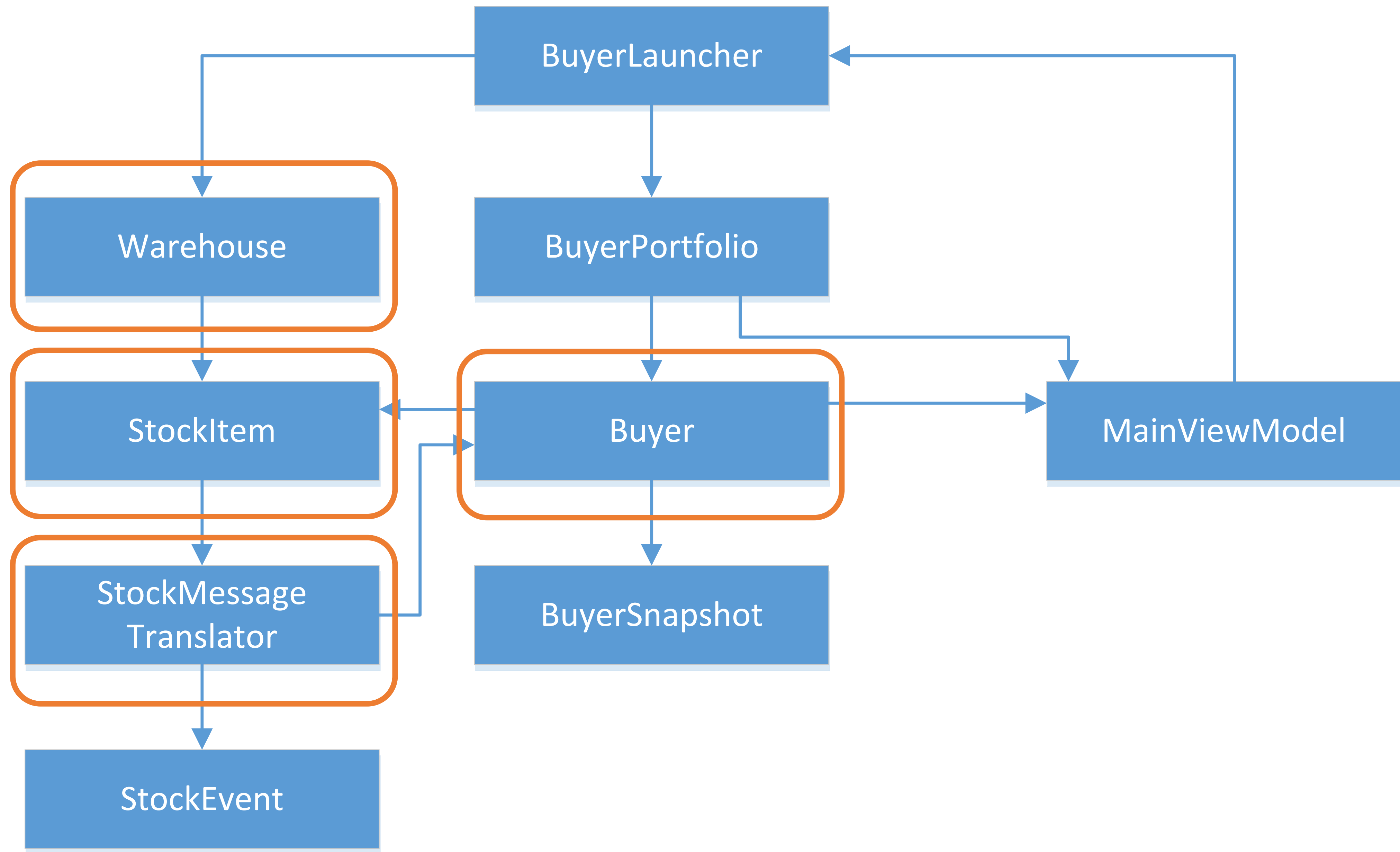
State-based testing

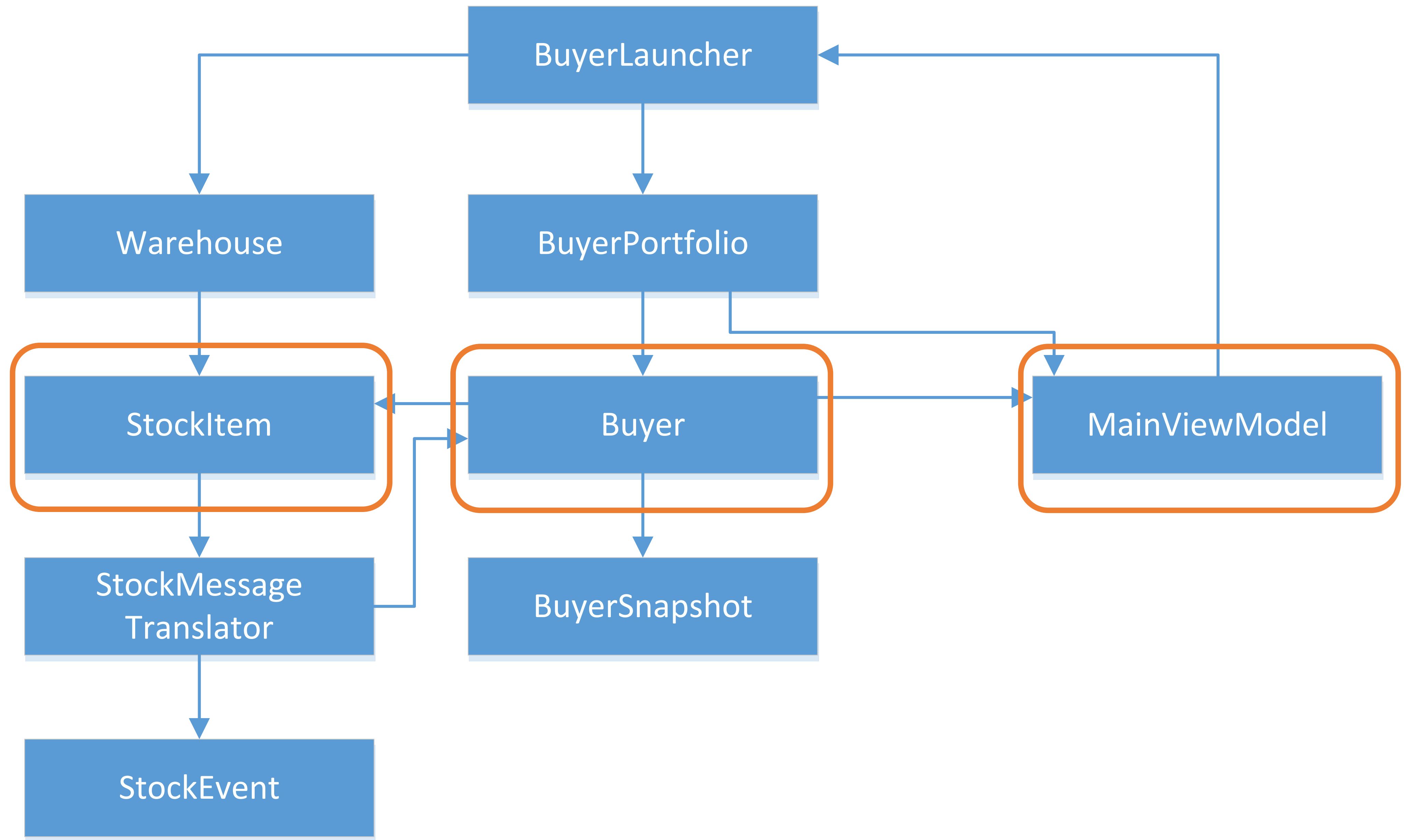
Collaboration-based testing

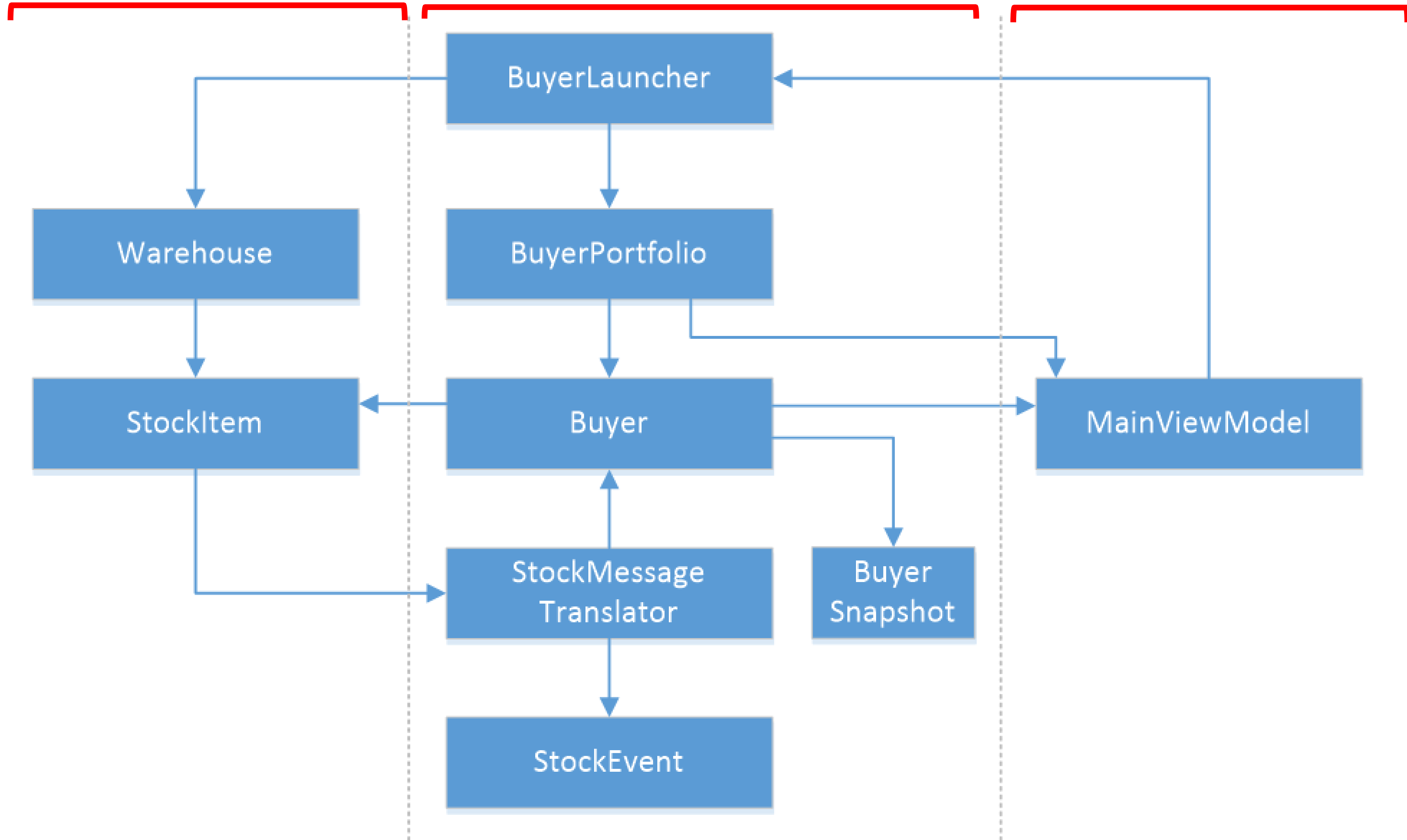
The further you take your tests away from the implementation details, the better.



Example


BuyerLauncher
Warehouse
BuyerPortfolio
StockItem
Buyer
MainViewModel
StockMessage
Translator
BuyerSnapshot
StockEvent


BuyerLauncher
Warehouse
BuyerPortfolio
StockItem
Buyer
MainViewModel
StockMessage
Translator
BuyerSnapshot
StockEvent


Communication with warehouse service
Domain model
Communication with user
BuyerLauncher
Warehouse
BuyerPortfolio
StockItem
Buyer
MainViewModel
StockMessage Translator
Buyer Snapshot
StockEvent

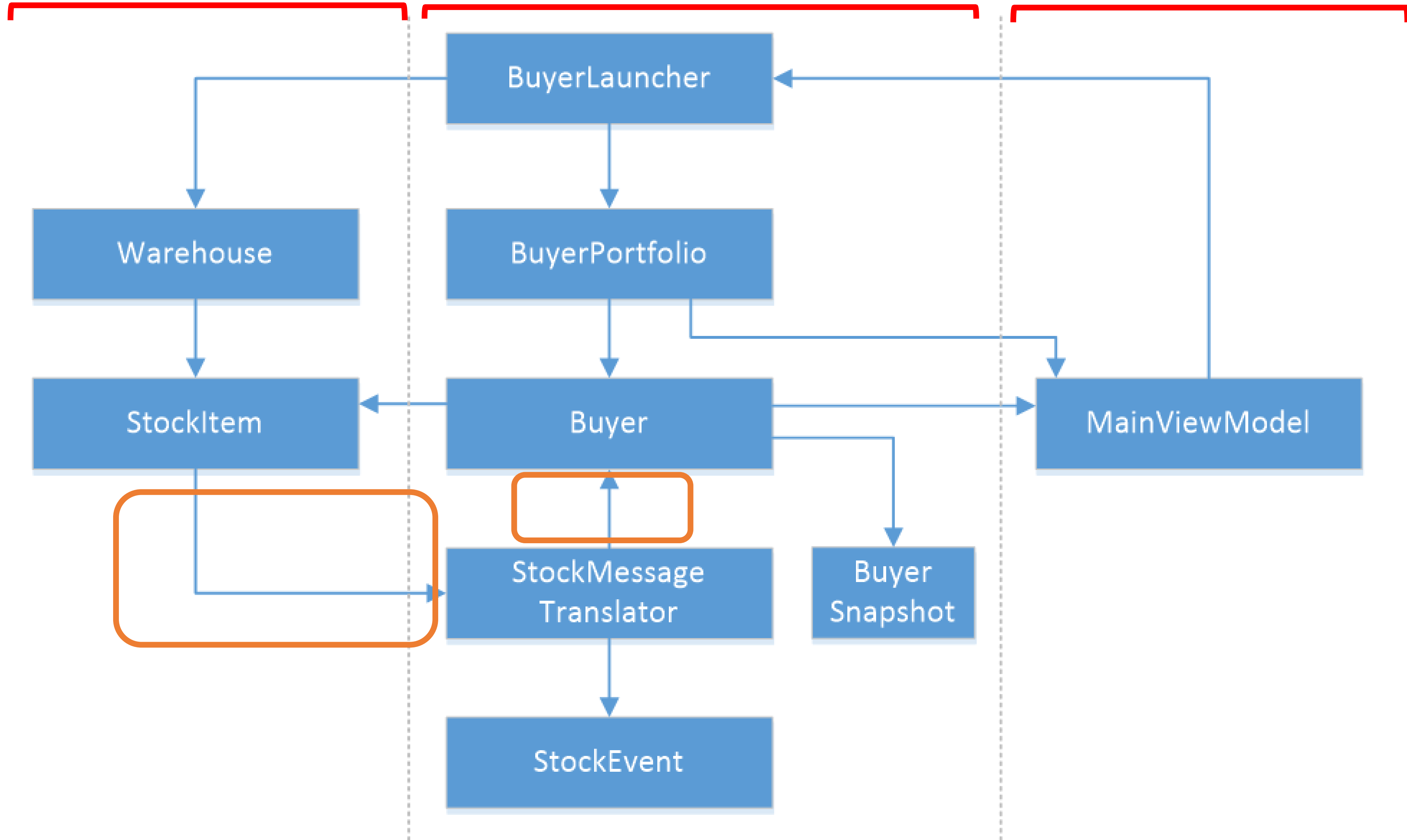
Unit Tests Analysis

```
[Fact]
public void Notifies_stock_closes_when_close_message_received()
{
    var sut = new StockMessageTranslator("Buyer");
    var mock = new Mock<IStockEventListener>();
    sut.AddStockEventListener(mock.Object);

    sut.ProcessMessage("Event: CLOSE;");

    mock.Verify(x => x.ItemClosed());
}
```

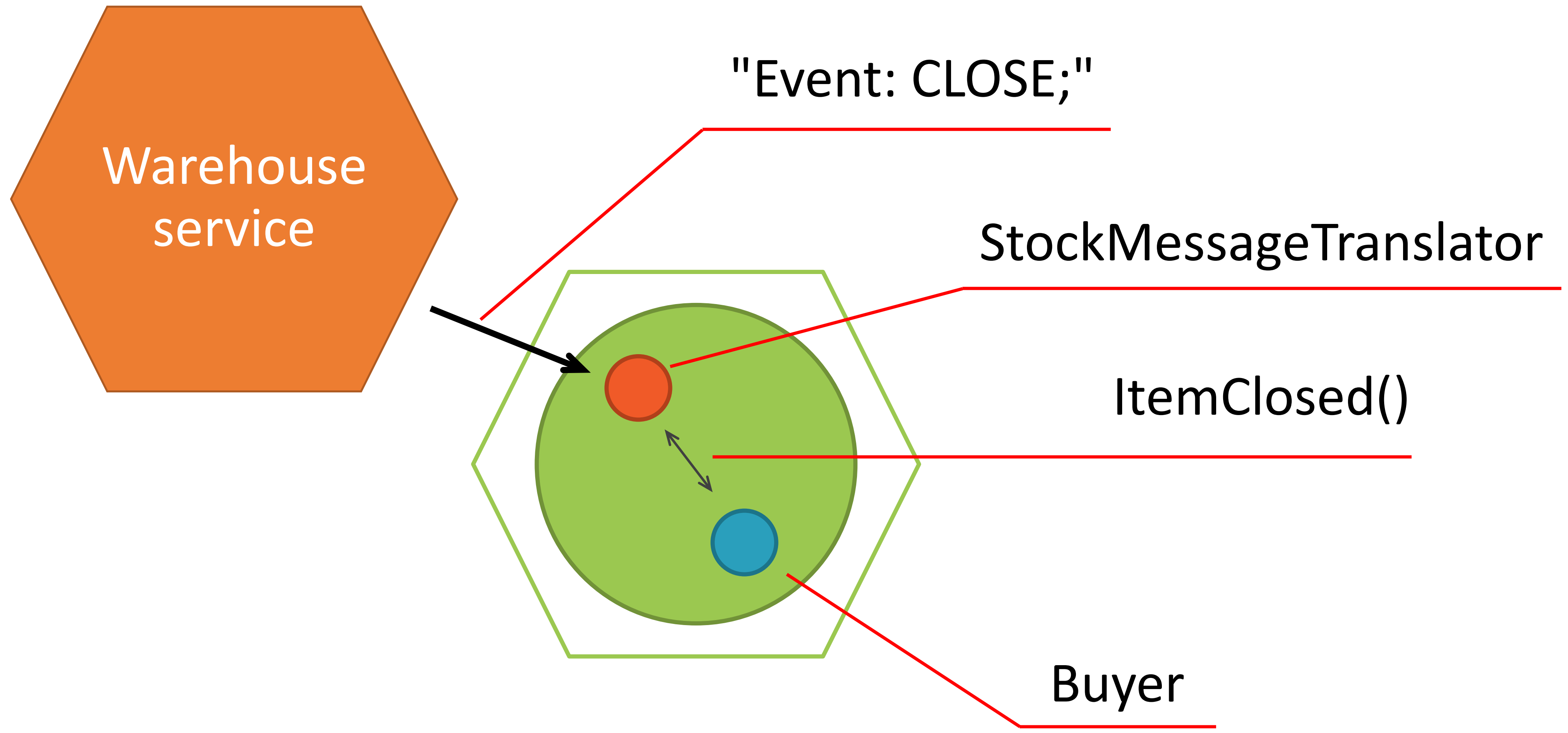
Communication with warehouse service

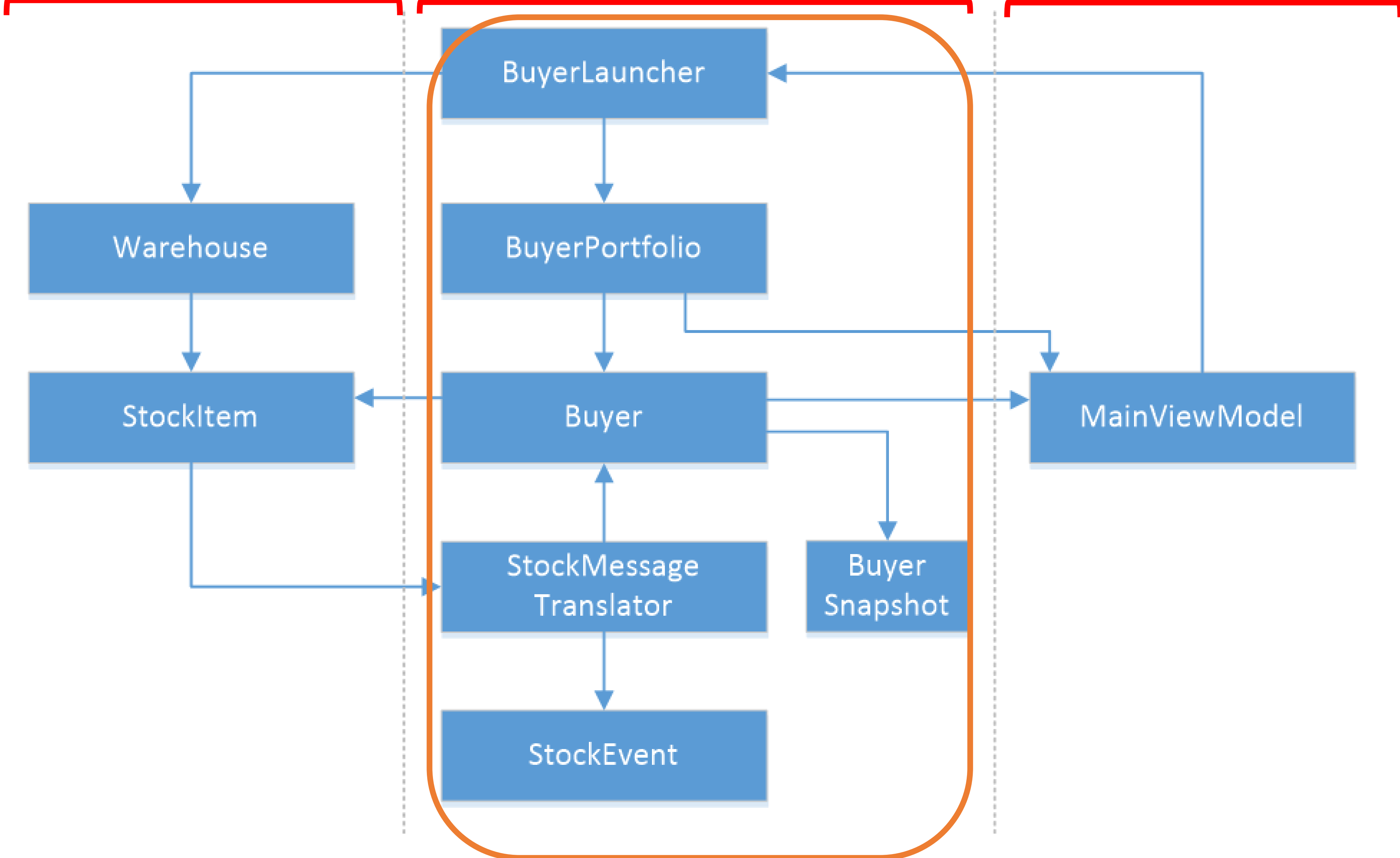
Domain model

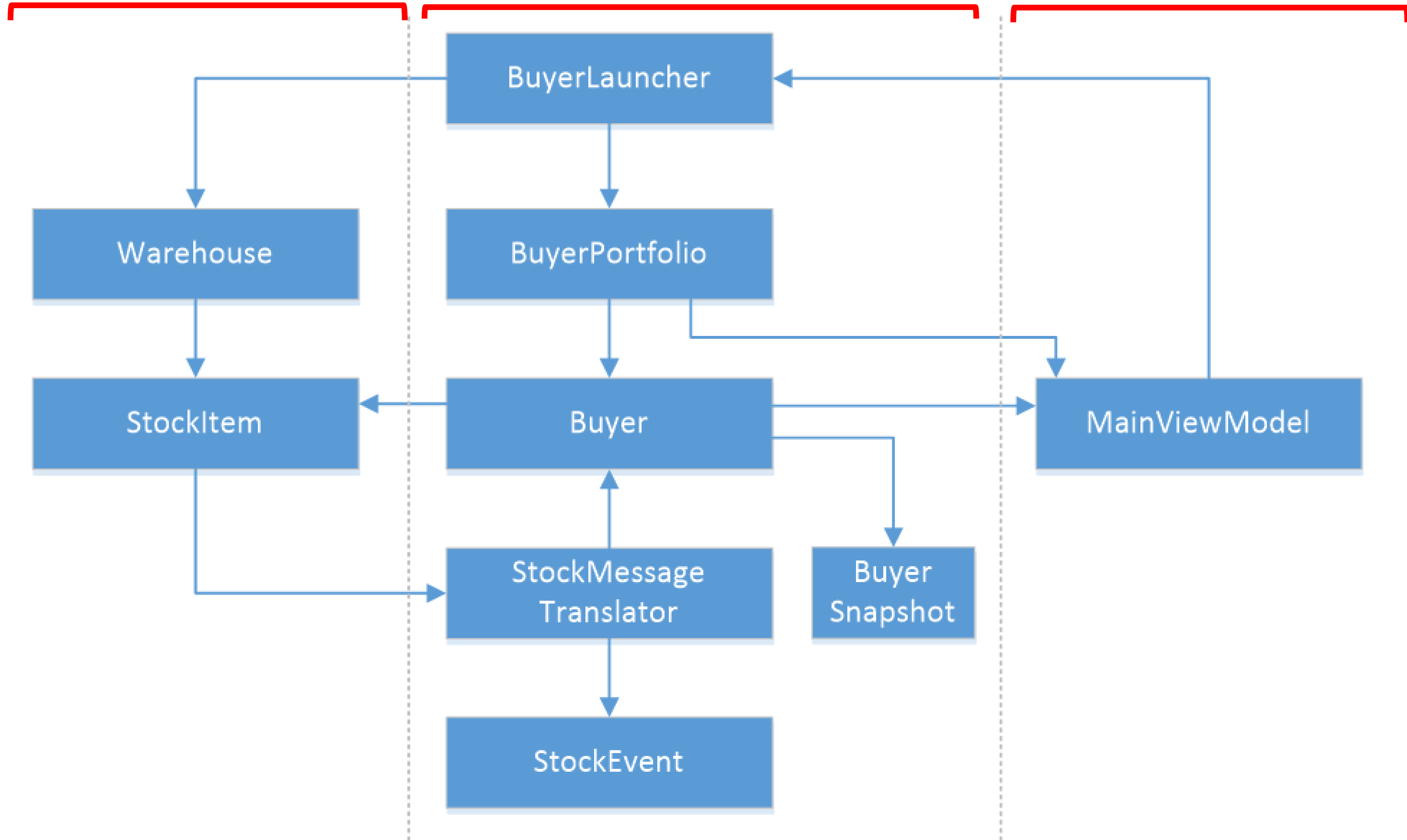
Communication with user



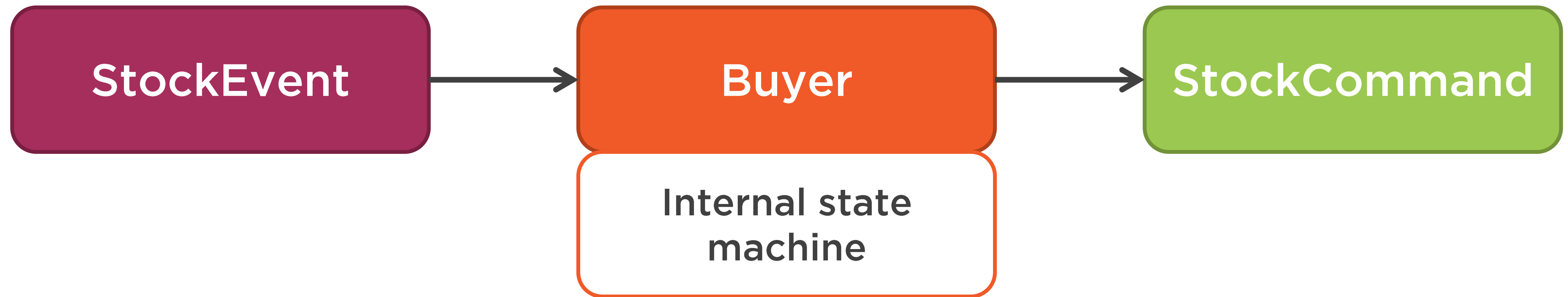
Collaboration-based Testing



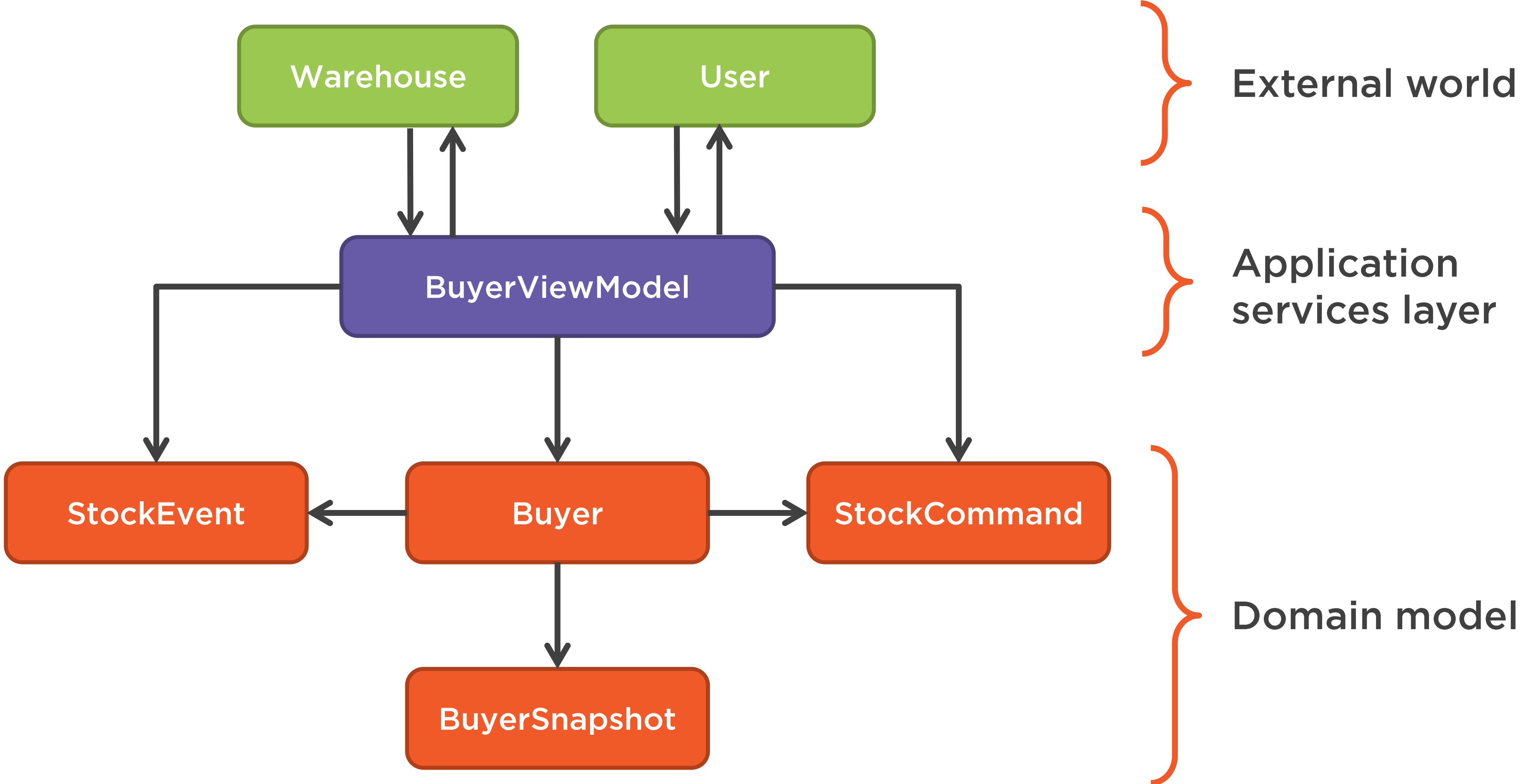

Communication with
warehouse service
Domain model
Communication with
user
BuyerLauncher
Warehouse
BuyerPortfolio
StockItem
Buyer
MainViewModel
StockMessage
Translator
Buyer
Snapshot
StockEvent


Communication with warehouse service
Domain model
Communication with user
BuyerLauncher
Warehouse
BuyerPortfolio
StockItem
Buyer
MainViewModel
StockMessage Translator
Buyer Snapshot
StockEvent

Refactored version



Refactored version



Unit Tests Analysis

```
[Fact]
public void Notifies_stock_closes_when_close_message_received()
{
    var sut = new StockMessageTranslator("Buyer");
    var mock = new Mock<IStockEventListener>();
    sut.AddStockEventListener(mock.Object);

    sut.ProcessMessage("Event: CLOSE;");

    mock.Verify(x => x.ItemClosed());
}
```

Unit Tests Analysis

```
[Fact]
public void Closes_when_item_closes()
{
    var buyer = CreateJoiningBuyer();

    StockCommand command = buyer.Process(StockEvent.Close());

    command.ShouldEqual(StockCommand.None());
    buyer.SnapshotShouldEqual(BuyerState.Closed, 0, 0, 0);
}
```

Unit Tests Analysis

```
[Fact]
public void Closes_when_item_closes()
{
    var buyer = CreateJoiningBuyer();

    StockCommand command = buyer.Process(StockEvent.Close());

    command.ShouldEqual(StockCommand.None());
    buyer.SnapshotShouldEqual(BuyerState.Closed, 0, 0, 0);
}
```

Resources

Growing Object-Oriented Software, Guided by Tests Without Mocks

- https://enterprisecraftsmanship.com/2016/07/05/growing-object-oriented-software-guided-by-tests-without-mocks/

When to use mocks

- https://enterprisecraftsmanship.com/2016/10/19/when-to-use-mocks/

Verifying collaborations at the system edges

- https://enterprisecraftsmanship.com/2016/10/26/2367/

Summary

- Components of a valuable test
  - Protection against regressions
  - Resistance to refactoring
  - Fast feedback
  - Maintainability
- Types of testing
  - Output-based testing
  - State-based testing
  - Collaboration-based testing
- Observable behavior vs Implementation detail

DOTNEXT

Pragmatic Unit Testing

Vladimir Khorikov

http://enterprisecraftsmanship.com

@vkhorikov