

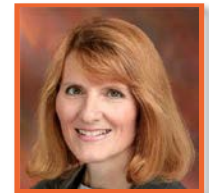
# Defending Your Methods Part 2: Validating Method Parameters

Deborah Kurata

<http://msmvps.com/blogs/deborahk/>

@DeborahKurata

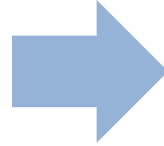
deborahk@insteptech.com



**pluralsight**   
hardcore dev and IT training

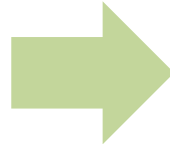
# Defensive Coding

Clean Code



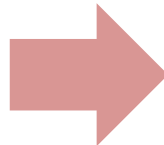
- Improves Comprehension
- Simplifies Maintenance
- Reduces Bugs

Testable Code  
+  
Unit Tests



- Improves Quality
- Confirms Maintenance
- Reduces Bugs

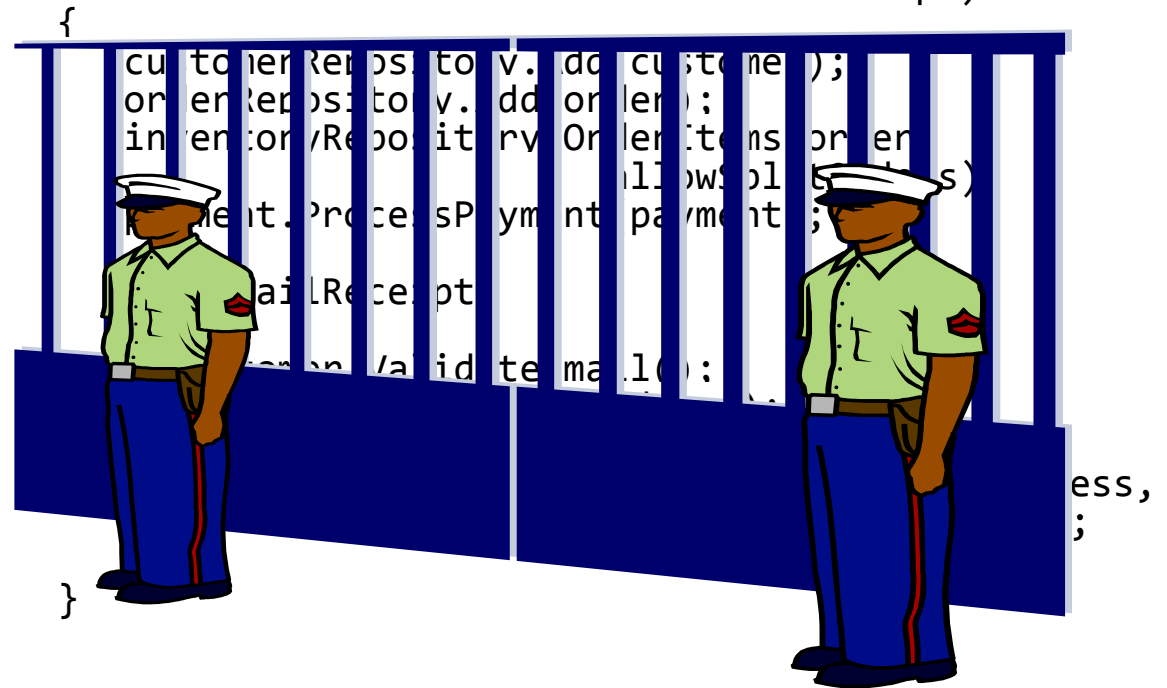
Validation  
+  
Exception Handling



- Improves Predictability
- More Consistent
- Reduces Bugs

# Defensive Coding

```
public void PlaceOrder(Customer customer,  
                        Order order,  
                        Payment payment,  
                        bool allowSplitOrders,  
                        bool emailReceipt)
```



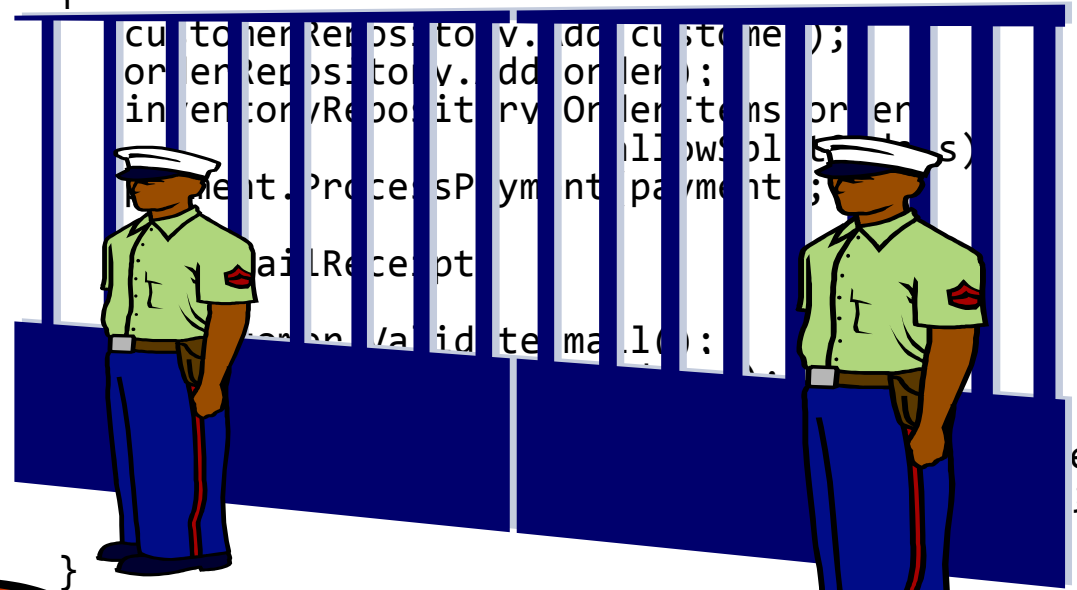
# Keep out the Garbage

```
public void PlaceOrder(Customer customer,  
                        Order order,  
                        Payment payment,  
                        bool allowSplitOrders,  
                        bool emailReceipt)
```

```
{
```

```
    customerRepository.Add(customer);  
    orderRepository.Add(order);  
    inventoryRepository.UpdateOrderItems(order);  
    paymentProcessor.ProcessPayment(payment);  
    emailReceiptService.SendEmail(customer, order, payment);  
}
```

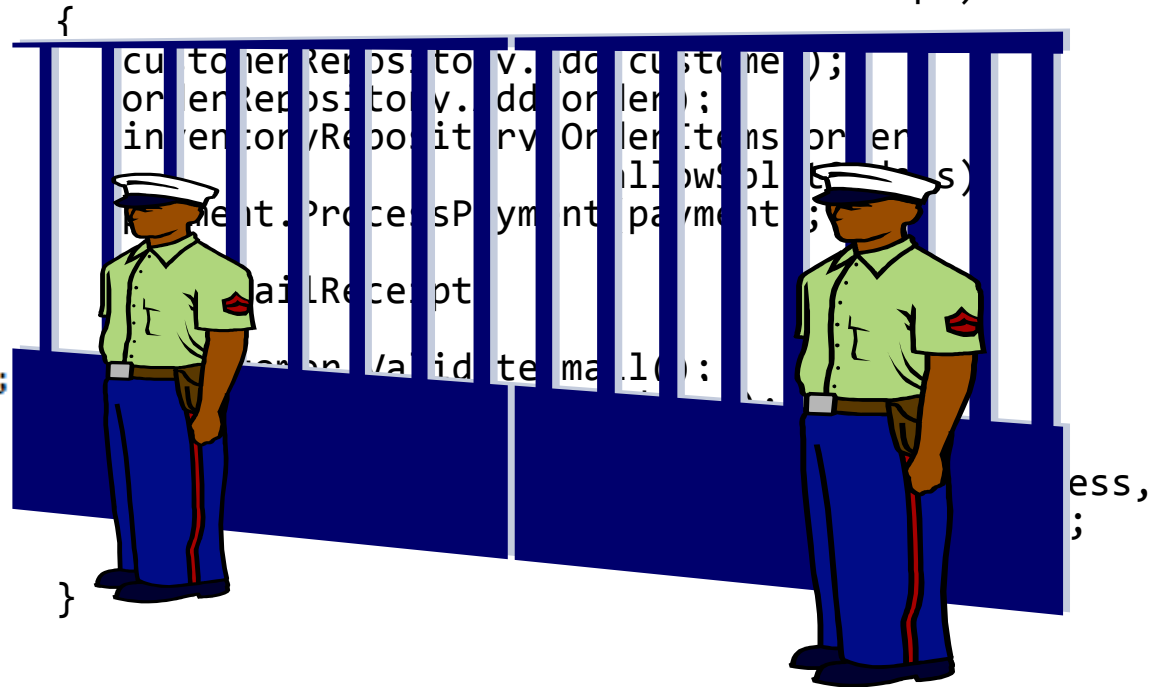
```
    }  
}
```



# Keep Out Invalid Parameters

```
public void PlaceOrder(Customer customer,  
                        Order order,  
                        Payment payment,  
                        bool allowSplitOrders,  
                        bool emailReceipt)
```


```
orderController.PlaceOrder(null,  
                            null,  
                            null,  
                            true,  
                            false);
```



# Allow Valid Parameters

```
public void PlaceOrder(Customer customer,
                        Order order,
                        Payment payment,
                        bool allowSplitOrders,
                        bool emailReceipt)
{
    customerRepository.Add(customer);
    orderRepository.Add(order);
    orderRepository.OrderItems(customerId,
                                allowSplitOrders);
    paymentProcessor.ProcessPayment(payment);
    emailLibrary.SendEmail(customer.Email,
                           "Here is your receipt",
                           emailReceipt);
}
```

orderController.PlaceOrder(validCustomer,  
validOrder,  
validPayment,  
emailReceipt:true,  
allowSplitOrders:false);



# Design by Contract

- What does a method expect?
- What does a method guarantee?
- What does the method maintain?

# Design by Contract

[INDIVIDUALS](#)[BUSINESS](#)[ACADEMIC](#)[FREE TRIAL](#)[BLOG](#)[SUPPORT](#)[deborahk89](#) ▼[Full Library](#)[Categories](#)[Authors](#)[Popular](#)[New Releases](#)

## Code Contracts

This course provides an introduction to Code Contracts in the Microsoft .NET Framework.

[g+1](#)[0](#)[Tweet](#)[2](#)[Like](#)[Share](#)[2](#)[Share](#)

Authored by: [John Sonmez](#)

Duration: 1h 51m

Level: Intermediate

Released: 7/17/2012

Features:

Course Rating: ★★★★★

[Table of Contents](#)[Description](#)[Transcript](#)[Exercise Files](#)[Assessment](#)[Discussion](#)

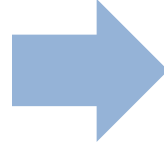
⊕ expand all | ⊖ collapse all

	Progress	Duration	
<a href="#">Code Contracts Overview</a>		00:14:04	
<a href="#">Basic Usage</a>		00:37:02	
<a href="#">Digging Deeper</a>		00:26:44	
<a href="#">Advanced</a>		00:33:27	



## Summary

Clean Code



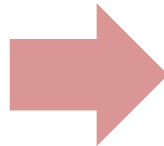
- Improves Comprehension
- Simplifies Maintenance
- Reduces Bugs

Testable Code  
+  
Unit Tests



- Improves Quality
- Confirms Maintenance
- Reduces Bugs

Validation  
+  
Exception Handling



- Improves Predictability
- More Consistent
- Reduces Bugs