Unit Test Craftsmanship

Gerard Meszaros

Independent Consultant

CTO of FeedXL.Com

singapore2016@gerardm.com

These Slides: http://singapore2016.xunitpatterns.com

My Background

- •Software developer
- •Development manager
- •Project Manager
- •Software architect

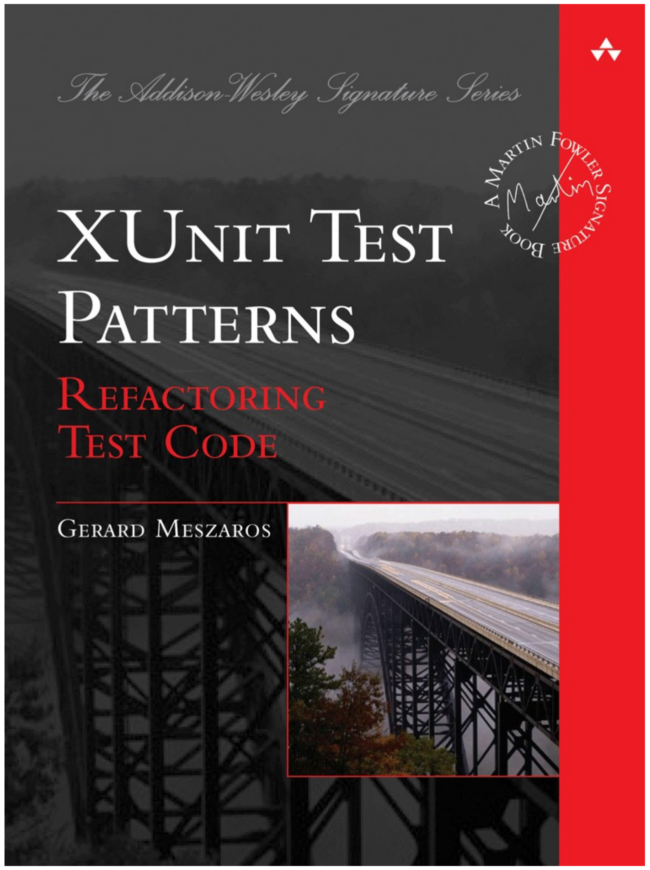
Embedded Telecom

- •OOA/OOD Mentor
- •Requirements (Use Case) Mentor
- •XP/TDD Mentor

I.T.

- •Agile PM Mentor
- •Test Automation Consultant & Trainer
- •Lean/Agile Coach/Consultant

Product & I.T.


The Addison-Wesley Signature Series
A Martin Fowler Signature Book
XUnit Test Patterns
Refactoring Test Code
Gerard Meszaros

Gerard Meszaros
xunit@gerardm.com

Customer Survey
What do you think about the quality of our offerings?
EXCELLENT
GOOD
AVERAGE
POOR
How would you rate our service?
EXCELLENT
GOOD
What are the ch
recommend us to others?
AVERAGE
POOR
EXCELLE

What Does it Take To be Successful?

Programming Experience

+ XUnit Experience

+ Testing experience

~~Robust Automated Tests~~

A Sobering Thought

Expect to have just as much test code as production code!

The Challenge: How To Prevent Doubling Cost of Software Maintenance?

Unit Test Automation Goals

- **Self Checking**
  - Test reports its own results; needs no human interpretation
- **Repeatable**
  - Test can be run many times in a row without human intervention
- **Robust**
  - Test produces same result now and forever
  - not affected by changes in the external environment
- **Complete**
  - Tests as Safety Net; verifies all component requirements
- **Maintainable**
  - Easy to understand; Tests as Documentation
- **Efficient**
  - Runs reasonably quickly
- **Specific**
  - Each test failure points to a specific piece of broken functionality – provides “defect triangulation”

Coding Objectives Comparison

	Production	*Testware*
Correctness	Important	Crucial
Maintainability	Important	Crucial
Execution Speed	Crucial	Somewhat
Reusability	Important	Not
Flexibility	Important	Not
Simplicity	Important?	Crucial
Ease of writing	Important?	Crucial
Obviousness	Not?	Crucial

Why are They so Crucial?

- **Tests need to be maintained along with rest of the software.**
- **Expect there to be as much testware as software.**
- **Testware must be much easier to maintain than the software, otherwise:**
  - It will slow you down
  - It will get left behind
  - Value drops to zero
  - You'll go back to manual testing

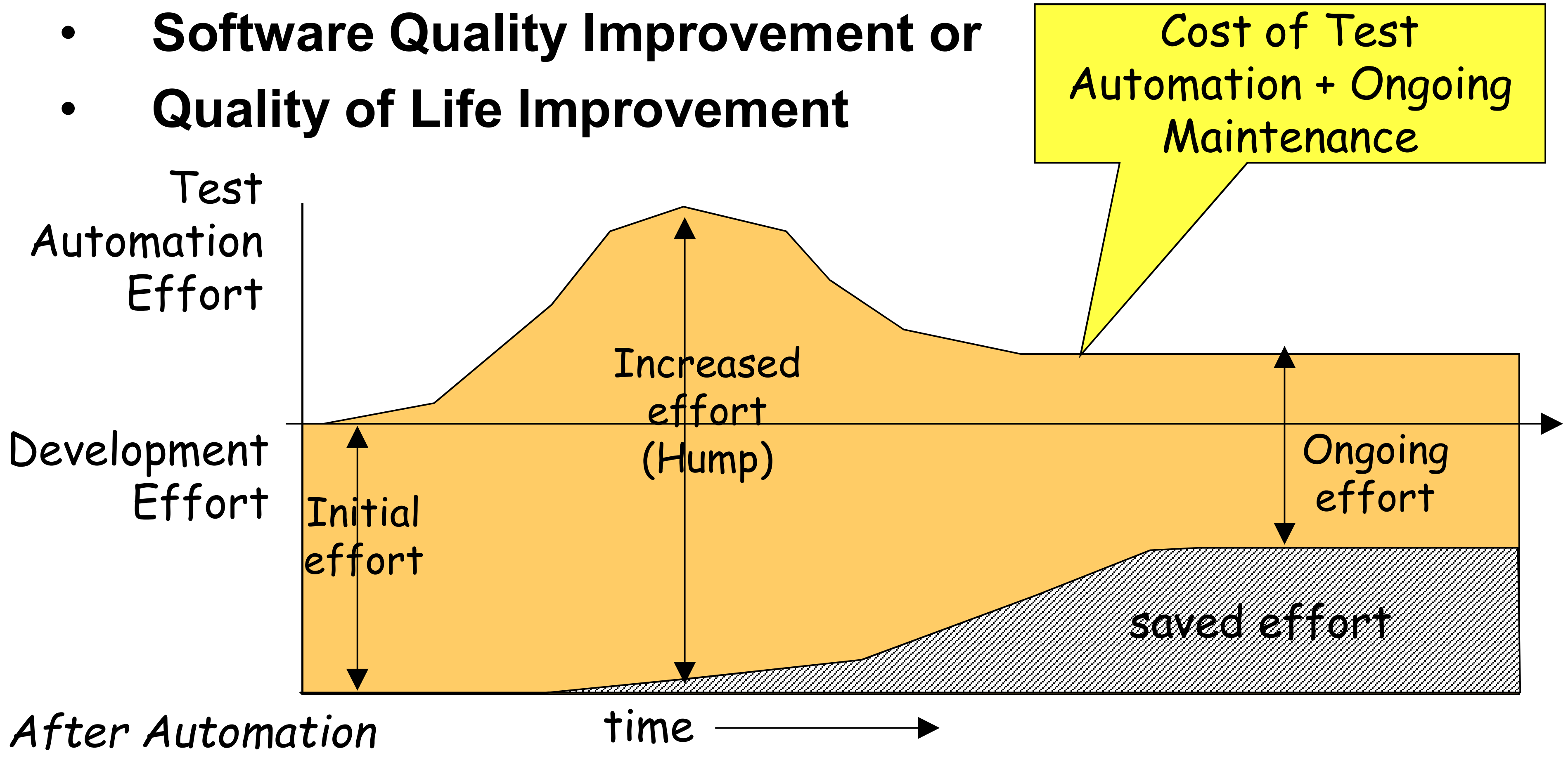
Critical Success Factor:

Writing tests in a maintainable style

Economics of Maintainability

Automated unit testing/checking is a lot easier to sell on

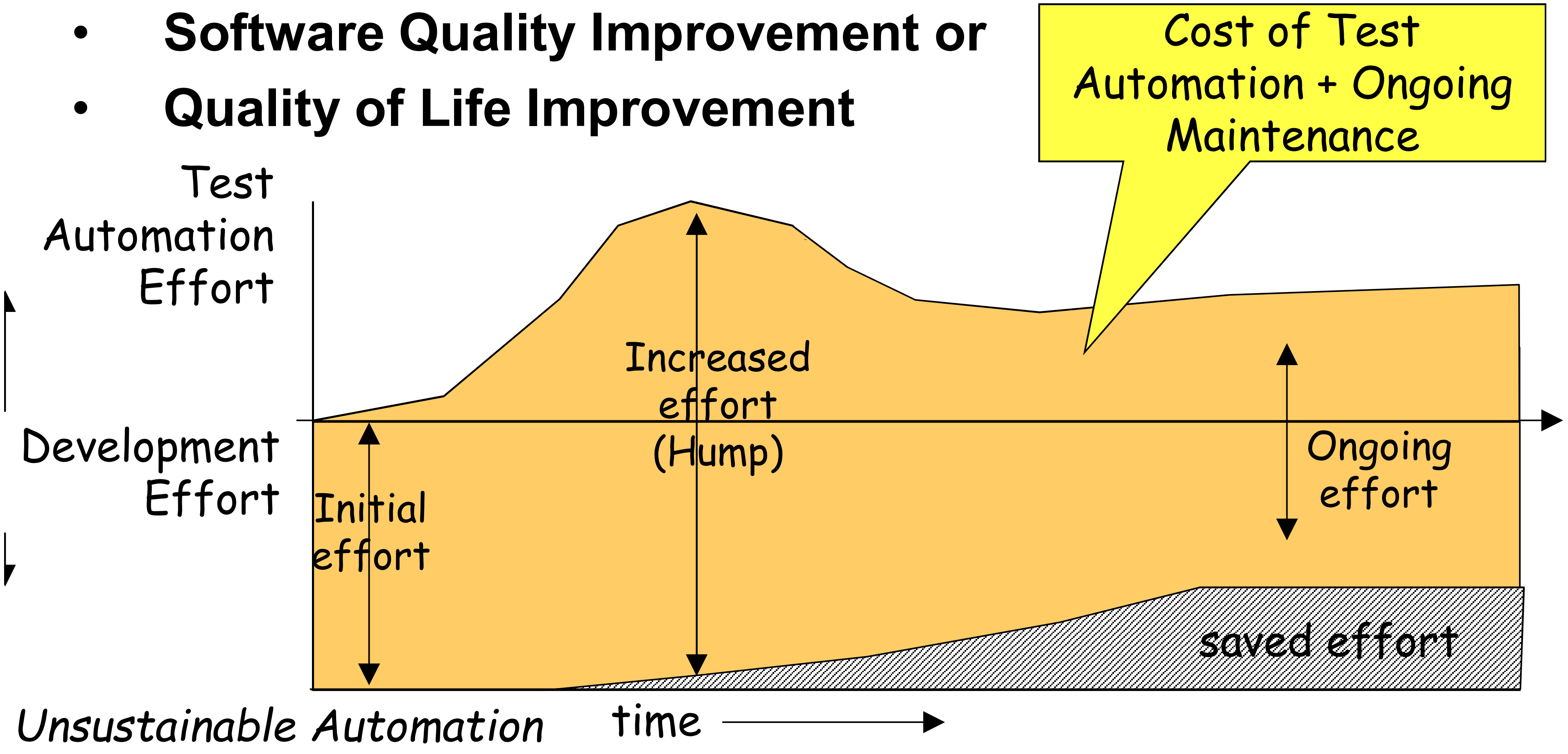
- **Cost reduction than**
- **Software Quality Improvement or**
- **Quality of Life Improvement**



Economics of Maintainability

Test Automation is a lot easier to sell on

- **Cost reduction than**
- **Software Quality Improvement or**
- **Quality of Life Improvement**

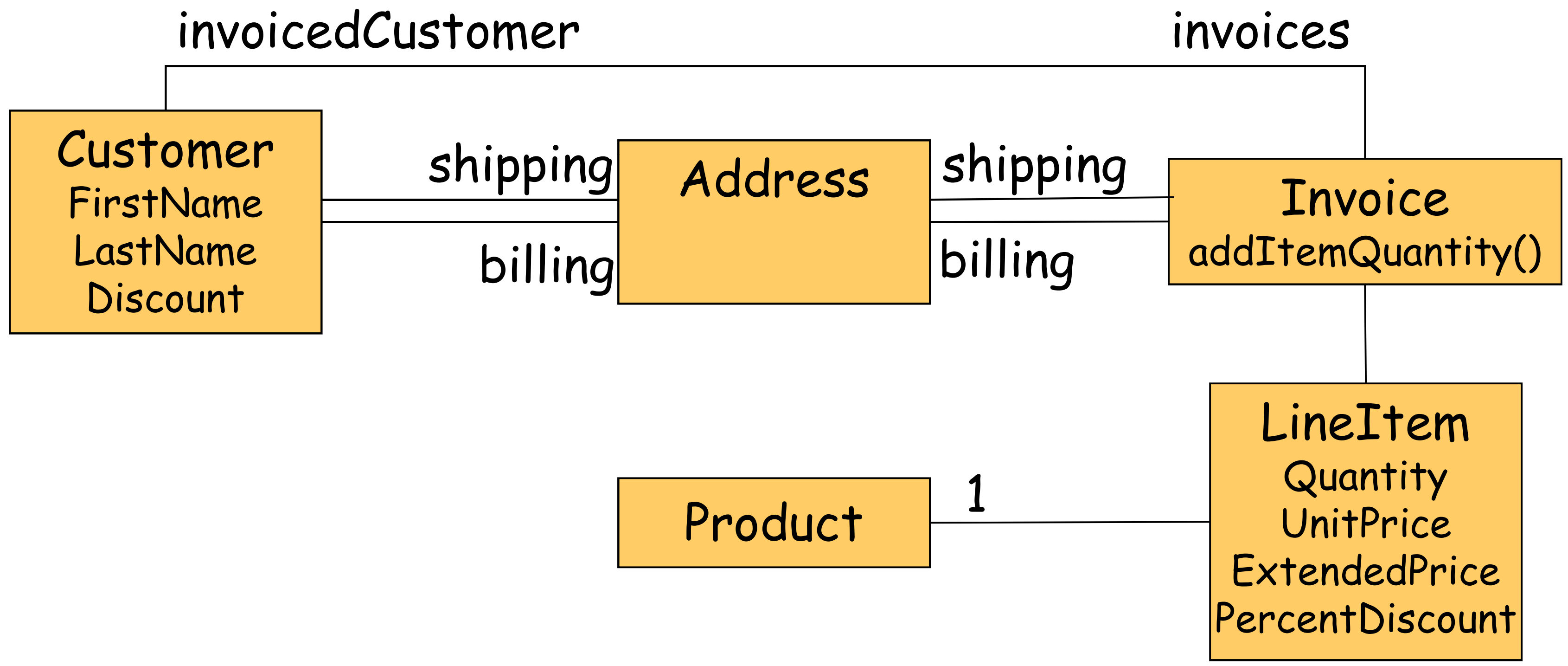


an example would be handy right about now

www.exampler.com

Example

- **Test addItemQuantity and removeLineItem methods of Invoice**



A Bunch of Tests / Checks:

```
TestInvoiceLineItems {
   testAddItemQuantity_singleQuantity()
   testAddItemQuantity_severalQuantity{..}
   testAddItemQuantity_duplicateProduct {..}
   testAddItemQuantity_differentProduct () {..}
   testAddItemQuantity_zeroQuantity {..}
   testAddItemQuantity_severalQuantity_... {..}
   testAddItemQuantity_discountedPrice_... {..}
   testRemoveItem_noItemsLeft… {..}
   testRemoveItem_oneItemLeft… {..}
   testRemoveItem_ severalItemsLeft… {..}
}
```

Do Your Tests Look Like:

```
public void testAddItemQuantity_severalQuantity () throws Exception {
  try {
     // Setup Fixture
     final int QUANTITY = 5;
     Address billingAddress = new Address("1222 1st St SW", "Calgary", "Alberta", "T2N 2V2",
           "Canada");
     Address shippingAddress = new Address("1333 1st St SW", "Calgary", "Alberta", "T2N 2V2",
           "Canada");
     Customer customer = new Customer(99, "John", "Doe", new BigDecimal("30"), billingAddress,
           shippingAddress);
     Product product = new Product(88, "SomeWidget", new BigDecimal("19.99"));
     Invoice invoice = new Invoice(customer);
     // Exercise SUT
     invoice.addItemQuantity(product, QUANTITY);
     // Verify Outcome
     List lineItems = invoice.getLineItems();
     if (lineItems.size() == 1) {
       LineItem actualLineItem = (LineItem)lineItems.get(0);
       assertEquals(invoice, actualLineItem.getInvoice());
       assertEquals(product, actualLineItem.getProduct());
       assertEquals(quantity, actualLineItem.getQuantity());
       assertEquals(new BigDecimal("30"), actualLineItem.getPercentDiscount());
       assertEquals(new BigDecimal("19.99"), actualLineItem.getUnitPrice());
       assertEquals(new BigDecimal(“69.96"), actualLineItem.getExtendedPrice());
     } else {
       assertTrue(“Invoice should have exactly one line item“, false);
     }
  } finally {
     deleteObject(expectedLineItem);
     deleteObject(invoice);
     deleteObject(product);
     deleteObject(customer);
     deleteObject(billingAddress);

}
```

You might be questioning their value!

How To Get To This?

```
@Test
public void addItemQuantity_severalQuantity () {
   QUANTITY = 5 ;
   product = givenAnyProduct();
   invoice = givenAnEmptyInvoice();

   invoice.addItemQuantity( product, QUANTITY);

   assertExactlyOneLineItem(
     invoice,
     expectedItem(
          invoice, product, QUANTITY,
               product.getPrice()* QUANTITY) );
}
```

The Whole Test

Given: ???

```
public void testAddItemQuantity_severalQuantity () throws Ex    tion {
    // Setup Fixture
    final int QUANTITY = 5;
    Address billingAddress = new Address("1222 1st St SW", "Calgary",
         "Alberta", "T2N 2V2", "Canada");
    Address shippingAddress = new Address("1333 1st St SW", "Calgary",
         "Alberta", "T2N 2V2", "Canada");
    Customer customer = new Customer(99, "John", "Doe", new BigDecimal("30"),
         billingAddress, shippingAddress);
    Product product = new Product(88, "SomeWidget", new Big
    Invoice invoice = new Invoice(customer);
    // Exercise SUT
    invoice.addItemQuantity(product, QUANTITY);
    // Verify Outcome
    List lineItems = invoice.getLineItems();
    if (lineItems.size() == 1) {
      LineItem actualLineItem = (LineItem)lineItems.get(0);
      assertEquals(invoice, actualLineItem.getInvoice());
      assertEquals(product, actualLineItem.getProduct());
      assertEquals(quantity, actualLineItem.getQuantity());
      assertEquals(new BigDecimal("30"), actualLineItem.getPercentDiscount());
      assertEquals(new BigDecimal("19.99"), actualLineItem.getUnitPrice());
      assertEquals(new BigDecimal("69.96"),
         actualLineItem.getExtendedPrice());
    } else {
      assertTrue("Invoice should have exactly one line item", false);
    }                      :
}
```

When we call addItemQuantity

Then: ???

Verifying the Outcome

```
List lineItems = invoice.getLineItems();
if (lineItems.size() == 1) {
  LineItem actualLineItem = (LineItem)lineItems.get(0);
  assertEquals(invoice, actualLineItem.getInvoice());
  assertEquals(product, actualLineItem.getProduct());
  assertEquals(quantity, actualLineItem.getQuantity());
  assertEquals(new BigDecimal("30"),
        actualLineItem.getPercentDiscount());
  assertEquals(new BigDecimal("19.99"),
        actualLineItem.getUnitPrice());
  assertEquals(new BigDecimal(“69.96"),
        actualLineItem.getExtendedPrice());
} else {
  assertTrue(“Invoice should have exactly one line item“,
                            false);
}
```

Obtuse Assertion

Use Better Assertion

```
List lineItems = invoice.getLineItems();
if (lineItems.size() == 1) {
  LineItem actualLineItem = (LineItem)lineItems.get(0);
  assertEquals(invoice, actualLineItem.getInvoice());
  assertEquals(product, actualLineItem.getProduct());
  assertEquals(quantity, actualLineItem.getQuantity());
  assertEquals(new BigDecimal("30"),
       actualLineItem.getPercentDiscount());
  assertEquals(new BigDecimal("19.99"),
       actualLineItem.getUnitPrice());
  assertEquals(new BigDecimal(“69.96"),
       actualLineItem.getExtendedPrice());
} else {
  fail("invoice should have exactly one line item");
}}
```

Use Better Assertion

```
List lineItems = invoice.getLineItems();
if (lineItems.size() == 1) {
  LineItem actualLineItem = (LineItem)lineItems.get(0);
  assertEquals(invoice, actualLineItem.getInvoice());
  assertEquals(product, actualLineItem.getProduct());
  assertEquals(quantity, actualLineItem.getQuantity());
  assertEquals(new BigDecimal("30"),
        actualLineItem.getPercentDiscount());
  assertEquals(new BigDecimal("19.99"),
        actualLineItem.getUnitPrice());
  assertEquals(new BigDecimal(“69.96"),
        actualLineItem.getExtendedPrice());
} else {
  fail("invoice should have exactly one line item");
}}
```

Hard-Wired Test Data

Fragile Tests

Expected Object

```
List lineItems = invoice.getLineItems();
if (lineItems.size() == 1) {
  LineItem actualLineItem = (LineItem)lineItems.get(0);
  LineItem expectedLineItem =
           newLineItem(invoice, product, QUANTITY);
  assertEquals(expectedLineItem.getInvoice(),
         actualLineItem.getInvoice());
  assertEquals(expectedLineItem.getProduct(),
         actualLineItem.getProduct());
  assertEquals(expectedLineItem.getQuantity(),
         actualLineItem.getQuantity());
  assertEquals(expectedLineItem.getPercentDiscount(),
         actualLineItem.getPercentDiscount());
  assertEquals(expectedLineItem.getUnitPrice(),
         actualLineItem.getUnitPrice());
  assertEquals(expectedLineItem.getExtendedPrice(),
         actualLineItem.getExtendedPrice());
} else {
  fail("invoice should have exactly one line item");
```

Expected Object

```
List lineItems = invoice.getLineItems();
if (lineItems.size() == 1) {
  LineItem actualLineItem = (LineItem)lineItems.get(0);
  LineItem expectedLineItem =  newLineItem(invoice,
         product, QUANTITY, product.getPrice()*QUANTITY );
  assertEquals(expectedLineItem.getInvoice(),
         actualLineItem.getInvoice());
  assertEquals(expectedLineItem.getProduct(),
         actualLineItem.getProduct());
  assertEquals(expectedLineItem.getQuantity(),
         actualLineItem.getQuantity());
  assertEquals(expectedLineItem.getPercentDiscount(),
         actualLineItem.getPercentDiscount());
  assertEquals(expectedLineItem.getUnitPrice(),
         actualLineItem.getUnitPrice());
  assertEquals(expectedLineItem.getExtendedPrice(),
         actualLineItem.getExtendedPrice());
} else {
  fail("invoice should have exactly one line item");
```

Verbose Test

Introduce Custom Assert

```
List lineItems = invoice.getLineItems();
if (lineItems.size() == 1) {
  LineItem actualLineItem = (LineItem)lineItems.get(0);
  LineItem expectedLineItem =  newLineItem(invoice,
         product, QUANTITY, product.getPrice()*QUANTITY );
  assertLineItemsEqual(expectedLineItem, actualLineItem);
```

Custom Assertion

```
} else {
  fail("invoice should have exactly one line item");
}
```

Introduce Custom Assert

```
List lineItems = invoice.getLineItems();
if (lineItems.size() == 1) {
   LineItem actualLineItem = (LineItem)lineItems.get(0);
   LineItem expectedLineItem = newLineItem(invoice,
            product, QUANTITY, product.getPrice()*QUANTITY );
   assertLineItemsEqual(expectedLineItem, actualLineItem);
} else {
   fail("invoice should have exactly one line item");
}
```

Conditional Test Logic

Replace Conditional Logic with Guard Assertion

```
List lineItems = invoice.getLineItems();
assertEquals("number of items",lineItems.size(),1);
LineItem actualLineItem = (LineItem)lineItems.get(0);
LineItem expectedLineItem =  newLineItem(invoice,
   product, QUANTITY, product.getPrice()*QUANTITY );
assertLineItemsEqual(expectedLineItem, actualLineItem);
```

The Whole Test

```
public void testAddItemQuantity_severalQuantity () throws Exception {
    // Setup Fixture
    final int QUANTITY = 5;
    Address billingAddress = new Address("1222 1st St SW", "Calgary",
        "Alberta", "T2N 2V2", "Canada");
    Address shippingAddress = new Address("1333 1st St SW",
        "Calgary", "Alberta", "T2N 2V2", "Canada");
    Customer customer = new Customer(99, "John", "Doe", new
        BigDecimal("30"), billingAddress, shippingAddress);
    Product product = new Product(88, "SomeWidget", new
        BigDecimal("19.99"));
    Invoice invoice = new Invoice(customer);
    // Exercise SUT
    invoice.addItemQuantity(product, QUANTITY);
    // Verify Outcome
    assertEquals("number of items",lineItems.size(),1);
    LineItem actualLineItem = (LineItem)lineItems.get(0);
    LineItem expectedLineItem = newLineItem(invoice, product,
        QUANTITY);
    assertLineItemsEqual(expectedLineItem, actualLineItem);
}
```

Code Smell

Hard-Coded Test Data

```
public void testAddItemQuantity_severalQuantity () {
    final int QUANTITY = 5;
    Address billingAddress = new Address("1222 1st St SW",
        "Calgary", "Alberta", "T2N 2V2", "Canada");

    Address shippingAddress = new Address("1333 1st St SW",
        "Calgary", "Alberta", "T2N 2V2", "Canada");

    Customer customer = new Customer(99, "John", "Doe", new
        BigDecimal("30"), billingAddress, shippingAddress);



    Product product = new Product(88, "SomeWidget
        BigDecimal("19.99"));

    Invoice invoice = new Invoice(customer);
    // Exercise SUT
    invoice.addItemQuantity(product, QUANTITY);
```

Hard-coded Test Data (Obscure Test)

Unrepeatable Tests

Distinct Generated Values

```
public void testAddItemQuantity_severalQuantity () {
    final int QUANTITY = 5 ;
    Address billingAddress = new Address(getUniqueString(),
        getUniqueString(), getUniqueString(),
        getUniqueString(), getUniqueString());
    Address shippingAddress = new Address(getUniqueString(),
        getUniqueString(), getUniqueString(),
        getUniqueString(), getUniqueString());
    Customer customer = new Customer(
        getUniqueInt(), getUniqueString(),
        getUniqueString(), getUniqueDiscount(),
        billingAddress, shippingAddress);
    Product product = new Product(
        getUniqueInt(), getUniqueString(),
        getUniqueNumber());
    Invoice invoice = new Invoice(customer);
```

Pattern

Distinct Generated Values

```
public void testAddItemQuantity_severalQuantity () {
   final int QUANTITY = 5 ;
   Address billingAddress = new Address(getUniqueString(),
       ~~getUniqueString(), getUniqueString()~~,
       ~~getUniqueString(), getUniqueString())~~;
   Address shippingAddress = new Address(getUniqueString(),
       ~~getUniqueString(), getUniqueString()~~,
       ~~getUniqueString(), getUniqueString())~~;
   Customer customer1 = new Customer(
       ~~getUniqueInt(), getUniqueS~~tring(),
       ~~getUniqueString(), getUnique~~Discount(),
       ~~billingAddress, shippingAddress);~~
   Product product = new Product(
       ~~getUniqueInt(), getUniqueString(),~~
       ~~getUniqueNumber());~~
   Invoice invoice = new Invoice(customer);
```

Irrelevant Information (Obscure Test)

Creation Method

```
public void testAddItemQuantity_severalQuantity () {
   final int QUANTITY = 5;
   Address billingAddress = createAnonymousAddress();


   Address shippingAddress = createAnonymousAddress();


   Customer customer = createCustomer( billingAddress,
       shippingAddress);


   Product product = createAnonymousProduct();


   Invoice invoice = new Invoice(customer);
```

Obscure Test - Irrelevant Information

```
public void testAddItemQuantity_severalQuantity () {
    final int QUANTITY = 5;
    ~~Address billingAddress = createAnonymousAddress()~~;
    ~~Address shippingAddress = createAnonymousAddress()~~;
    Customer customer = createCustomer(
        ~~billingAddress,~~ ~~shippingAddress~~);
    Product product = createAnonymousProduct();
    Invoice invoice = new Invoice(customer);
    // Exercise
    invoice.addItemQuantity(product, QUANTITY);
    // Verify
    LineItem expectedLineItem =    newLineItem(invoice,
        product, QUANTITY, product.getPrice()*QUANTITY );
    List lineItems = invoice.getLineItems();
    assertEquals("number of items", lineItems.size(), 1);
    LineItem actualLineItem = (LineItem)lineItems.get(0);
    assertLineItemsEqual(expectedLineItem, actualLineItem);
}
```

Irrelevant Information (Obscure Test)

Remove Irrelevant Information

```
public void testAddItemQuantity_severalQuantity () {
    final int QUANTITY = 5 ;


    ~~Customer customer = createAnonymousCustomer()~~;

    Product product = createAnonymousProduct();
    Invoice invoice = new Invoice(~~customer~~);
    // Exercise
    invoice.addItemQuantity(product, QUANTITY);
    // Verify
    LineItem expectedLineItem =    newLineItem(invoice,
        product, QUANTITY, product.getPrice()*QUANTITY );
    List lineItems = invoice.getLineItems();
    assertEquals("number of items", lineItems.size(), 1);
    LineItem actualLineItem = (LineItem)lineItems.get(0);
    assertLineItemsEqual(expectedLineItem, actualLineItem);
}
```

Irrelevant Information (Obscure Test)

Remove Irrelevant Information

```
public void testAddItemQuantity_severalQuantity () {
    final int QUANTITY = 5 ;




    Product product = createAnonymousProduct();
    Invoice invoice = createAnonymousInvoice()
    // Exercise
    invoice.addItemQuantity(product, QUANTITY);
    // Verify
    LineItem expectedLineItem =    newLineItem(invoice,
        product, QUANTITY, product.getPrice()*QUANTITY );
    List lineItems = invoice.getLineItems();
    assertEquals("number of items", lineItems.size(), 1);
    LineItem actualLineItem = (LineItem)lineItems.get(0);
    assertLineItemsEqual(expectedLineItem, actualLineItem);
}
```

Introduce Custom Assertion

```
public void testAddItemQuantity_severalQuantity () {
    final int QUANTITY = 5 ;




    Product product = createAnonymousProduct();
    Invoice invoice = createAnonymousInvoice()
    // Exercise
    invoice.addItemQuantity(product, QUANTITY);
    // Verify
    LineItem expectedLineItem =    newLineItem(invoice,
        product, QUANTITY, product.getPrice()*QUANTITY );
    ~~List lineItems = invoice.getLineItems~~();
    ~~assertEquals("number of items", lineItems.size(), 1);~~
    ~~LineItem actualLineItem = (LineItem)lineItems.get(0);~~
    assertLineItemsEqual(expectedLineItem, actualLineItem);
}
```

Mechanics hides Intent

Introduce Custom Assertion

```
public void testAddItemQuantity_severalQuantity () {
   final int QUANTITY = 5 ;




   Product product = createAnonymousProduct();
   Invoice invoice = createAnonymousInvoice()
   // Exercise
   invoice.addItemQuantity(product, QUANTITY);
   // Verify
   LineItem expectedLineItem =   newLineItem(invoice,
      product, QUANTITY, product.getPrice()*QUANTITY );
   assertExactlyOneLineItem(invoice, expectedLineItem );

}
```

The Whole Test – Done

```
public void testAddItemQuantity_severalQuantity () {
   // Setup
   final int QUANTITY = 5 ;
   Product product = createAnonymousProduct();
   Invoice invoice = createAnonymousInvoice();
   // Exercise
   invoice.addItemQuantity(product, QUANTITY);
   // Verify
   LineItem expectedLineItem =   newLineItem(invoice,
       product, QUANTITY, product.getPrice()*QUANTITY );
   assertExactlyOneLineItem(invoice, expectedLineItem );
}
```

Four-Phase Test

```
public void testAddItemQuantity_severalQuantity () {
   // Setup     or  // Arrange
   final int QUANTITY = 5 ;
   Product product = createAnonymousProduct();
   Invoice invoice = createAnonymousInvoice();
   // Exercise or // Act
   invoice.addItemQuantity(product, QUANTITY);
   // Verify   or // Assert
   LineItem expectedLineItem =    newLineItem(invoice,
       product, QUANTITY, product.getPrice()*QUANTITY );
   assertExactlyOneLineItem(invoice, expectedLineItem );
}  // Teardown
     // Shouldn't be needed
```

- Bill Wake http://xp123.com/articles/3a-arrange-act-assert/

This terminology reinforces our focus on mechanics, not intent!

Four-Phase Test

```
public void testAddItemQuantity_severalQuantity () {
   // Setup     or  // Arrange
   final int QUANTITY = 5 ;
   Product product = createAnonymousProduct();
   Invoice invoice = createAnonymousInvoice();
   // Exercise  or // Act
   invoice.addItemQuantity(product, QUANTITY);
   // Verify    or // Assert
   LineItem expectedLineItem =   newLineItem(invoice,
       product, QUANTITY, product.getPrice()*QUANTITY );
   assertExactlyOneLineItem(invoice, expectedLineItem );
}
```

Given an empty invoice

when I call addItemQuantity

Then the invoice will end up with exactly 1 lineItem on it.

- Use Domain-Specific Language
- Say Only What is Relevant

Improving Terminology

```
public void testAddItemQuantity_severalQuantity () {
   // Given
   final int QUANTITY = 5 ;
   Product product = createAnonymousProduct();
   Invoice invoice = createAnonymousInvoice();
   // When
   invoice.addItemQuantity(product, QUANTITY);
   // Then
   LineItem expectedLineItem =    newLineItem(invoice,product,
       QUANTITY, product.getPrice()*QUANTITY );
   assertExactlyOneLineItem(invoice, expectedLineItem );
}
```

- Use Domain-Specific Language
- Say Only What is Relevant

Improving Terminology

```
@Test public void
testAddItemQuantity_severalQuantity () {
   final int QUANTITY = 5 ;
   Product product = createAnonymousProduct();
   Invoice invoice = createAnonymousInvoice();
   // When
   invoice.addItemQuantity(product, QUANTITY);
   // Then
   LineItem expectedLineItem =    newLineItem(invoice, product,
       QUANTITY, product.getPrice()*QUANTITY );
   assertExactlyOneLineItem(invoice, expectedLineItem );
}
```

- Use Domain-Specific Language
- Say Only What is Relevant

Rename

Improving Terminology

```
@Test public void
addItem_severalQuantity_itemValueIsQuantityTimesProductPrice(){
   final int QUANTITY = 5 ;
   Product product = createAnonymousProduct();
   Invoice invoice = createAnonymousInvoice();
   // When
   invoice.addItemQuantity(product, QUANTITY);
   // Then
   LineItem expectedLineItem =    newLineItem(invoice, product,
       QUANTITY, product.getPrice()*QUANTITY );
   assertExactlyOneLineItem(invoice, expectedLineItem );
}
```

Constantly Strive to Improve Readability

- Use Domain-Specific Language
- Say Only What is Relevant

Rename, Inline Local, Rename

Improving Terminology

```
@Test public void
addItem_severalQuantity_itemValueIsQuantityTimesProductPrice(){
   final int QUANTITY = 5 ;
   Product product = createAnonymousProduct();
   Invoice invoice = createAnonymousInvoice();
   // When
   invoice.addItemQuantity(product, QUANTITY);
   // Then
   shouldBeExactlyOneLineItemOn(invoice,
       expectedLineItem(invoice, product, QUANTITY,
                        product.getPrice()*QUANTITY)  );
}
```

Constantly Strive to Improve Readability

- Use Domain-Specific Language
- Say Only What is Relevant

Another Rename

Improving Terminology

```
@Test public void
addItem_severalQuantity_itemValueIsQuantityTimesProductPrice(){
   final int QUANTITY = 5 ;
   Product product = createIrrelevantProduct();
   Invoice invoice = createIrrelevantInvoice();
   // When
   invoice.addItemQuantity(product, QUANTITY);
   // Then
   shouldBeExactlyOneLineItemOn(invoice,
       expectedLineItem(invoice, product, QUANTITY,
                        product.getPrice()*QUANTITY)  );
}
```

Constantly Strive to Improve Readability

- Use Domain-Specific Language
- Say Only What is Relevant

Improving Terminology

```
@Test public void
addItem_severalQuantity_itemValueIsQuantityTimesProductPrice(){
   final int QUANTITY = 5 ;
   Product product = givenAnyProduct();
   Invoice invoice = givenAnEmptyInvoice();
   // When
   invoice.addItemQuantity(product, QUANTITY);
   // Then
   shouldBeExactlyOneLineItemOn(invoice,
       expectedLineItem(invoice, product, QUANTITY,
                        product.getPrice()*QUANTITY)  );
}
```

Naming as a Process - Arlo Belshee

- Use Domain-Specific Language
- Say Only What is Relevant

Test Coverage

```
TestInvoiceLineItems {
   testAddItemQuantity_singleQuantity()
   testAddItemQuantity_severalQuantity{..}
   testAddItemQuantity_duplicateProduct {..}
   testAddItemQuantity_differentProduct () {..}
   testAddItemQuantity_zeroQuantity {..}
   testAddItemQuantity_severalQuantity_... {..}
   testAddItemQuantity_discountedPrice_... {..}
   testRemoveItem_noItemsLeft… {..}
   testRemoveItem_oneItemLeft… {..}
   testRemoveItem_ severalItemsLeft… {..}
}
```

A whole bunch of Renames

Test Coverage

```
TestInvoiceLineItems {
  addItem_singleQuantity_itemValueIsProductPrice
  addItem_severalQuantity_itemValueIsQuantityTimesPr…
  addItem_duplicateProduct_singleItemHasSumOfQuantity
  addItem_differentProduct_oneItemPerProduct
  addItem_zeroQuantity_noItemAdded
  addItem_customerWithDiscount_itemValueIsDiscounted
  removeItem_onlyItem_noItemsLeft…
  removeItem_severalItems_oneLessItemLeft
  removeItem_severalItems_severalItemsLeft
}
```

Test Coverage

```
TestInvoiceLineItems {
   addItem_singleQuantity_itemValueIsProductPrice
   addItem_severalQuantity_itemValueIsQuantityTimesPr…
   addItem_duplicateProduct_singleItemHasSumOfQuantity
   addItem_differentProduct_oneItemPerProduct
   addItem_zeroQuantity_noItemAdded
   addItem_customerWithDiscount_itemValueIsDiscounted
   removeItem_onlyItem_noItemsLeft…
   removeItem_severalItems_oneLessItemLeft
   removeItem_severalItems_severalItemsLeft
}
```

GGM53

Rapid Test Writing

```
@Test public void
addItem_duplicateProduct_singleItemHasSumOfQuantities () {
   final int QUANTITY = 1 ;
   final int QUANTITY2 = 2 ;
   Product product = givenAnyProduct();
   Invoice invoice = givenAnEmptyInvoice();
   // When
   invoice.addItemQuantity(product, QUANTITY);
   invoice.addItemQuantity(product, QUANTITY2);
   // Then
   shouldBeExactlyOneLineItemOn(invoice,
        expectedLineItem(invoice, product, QUANTITY+QUANTITY2,
            product.getPrice() * (QUANTITY+QUANTITY2) );
}
```

Given an empty invoice

when I call addItemQuantity twice with same product

The invoice will end up with exactly 1 lineItem on it for the sum of the two calls to add..().

Slide 47

GGM53 Redo using new naming conventions
Gerard Meszaros, 12/10/19

Test Coverage

```
TestInvoiceLineItems {
   addItem_singleQuantity_itemValueIsProductPrice…{..}
   addItem_severalQuantity_itemValueIsQuantityTi… {..}
   addItem_duplicateProduct_singleItemHasSumOfQ...{..}
   addItem_differentProduct_oneItemPerProduct( ) {..}
   addItem_zeroQuantity_noItem… {..}
   addItem_severalQuantity_... {..}
   addItem_discountedPrice_... {..}
   removeItem_noItemsLeft… {..}
   removeItem_oneItemLeft… {..}
   removeItem_ severalItemsLeft… {..}
}
```

Rapid Test Writing

```
@Test public void
addItem_differentProduct_oneItemPerProduct() {
   final int QUANTITY = 1;

   Product product1 = givenAnyProduct();

   Invoice invoice = givenAnEmptyInvoice();
   // When
   invoice.addItemQuantity(product1, QUANTITY);

   // Then
   shouldBeExactlyTwoLineItems(invoice,
        expectedLineItem(invoice, product1, QUANTITY,
              product1.getPrice() * QUANTITY1)
        expectedLineItem(invoice, product2, QUANTITY2,
              product2.getPrice() * QUANTITY2 )    );
  }
```

Rapid Test Writing

```
@Test public void
addItem_differentProduct_oneItemPerProduct() {
    final int QUANTITY = 1;
    final int QUANTITY2 = 2;
    Product product1 = givenAnyProduct();
    Product product2 = givenAnyProduct();
    Invoice invoice = givenAnEmptyInvoice();
    // When
    invoice.addItemQuantity(product1, QUANTITY);
    invoice.addItemQuantity(product2, QUANTITY2);
    // Then
   shouldBeExactlyTwoLineItems(invoice,
        expectedLineItem(invoice, product1, QUANTITY,
               product1.getPrice() * QUANTITY1)
        expectedLineItem(invoice, product2, QUANTITY2,
               product2.getPrice() * QUANTITY2 )    );
  }
```

Given an empty invoice

when I call addItemQuantity twice with different products

The invoice will end up with 2 lineItems on it, one for each of the two calls to add..().

Removing Deodorant

```
@Test public void
addItem_differentProduct_oneItemPerProduct () {
   final int QUANTITY = 1;
   final int QUANTITY2 = 2;
   Product product1 = givenAnyProduct();
   Product product2 = givenAnyProduct();
   Invoice invoice = givenAnEmptyInvoice();

   invoice.addItemQuantity(product1, QUANTITY);
   invoice.addItemQuantity(product2, QUANTITY2);

   shouldBeExactlyTwoLineItems(invoice,
        expectedLineItem(invoice, product1, QUANTITY,
              product1.getPrice() * QUANTITY1)
        expectedLineItem(invoice, product2, QUANTITY2,
              product2.getPrice() * QUANTITY2 )    );
}
```

Given an empty invoice

when I call addItemQuantity twice with different products

The invoice will end up with 2 lineItems on it, one for each of the two calls to add..().

Benefits

- **Writing tests is faster**
  - Less code to write
- **Reading tests is faster, too.**
  - Less code to read
- **Much easier to see what's different from one test to another.**
  - Differences are fairly obvious
- **Tests are much less fragile**
  - Most code breakages are in test utility methods, not the tests themselves.

Nice, But Couldn't We Avoid the Refactoring?

Reducing the Need to Refactor Tests

```
@Test
public void generateInvoice_should…() throws Ex… {
    // setup and exercise omitted
    // verify the actual invoice header matches the expected header
    assertNotNull(“Number”, newInvoice.getNumber());
    assertEquals(“Name”, account. getName(), newInvoice.getName());
    assertEquals(“Address”, account. getAddr(), newInvoice.getAddr());
    assertEquals(“City”, account. getC
```

Reducing the Need to Refactor Tests

```
@Test
public void generateInvoice_should…() throws Ex… {
    // setup and exercise omitted
    assertInvoiceHeaderIs( newInvoice , expectedHeader(account) );
    shouldBeExactlyTwoLineItemsOn(
                invoice,
                expectedLineItem( invoice, product1, QUANTITY,
                                  product1.getPrice() * QUANTITY1)
                expectedLineItem( invoice, product2, QUANTITY2,
                                  product2.getPrice() * QUANTITY2)
                );

}
```

That's Better!

Now, All I have to do is implement these test utility methods (test-driven, of course!)

What Does it Take To be Successful?

Programming Experience

+ XUnit Experience

+ Testing experience

+ Good naming

+Regular refactoring

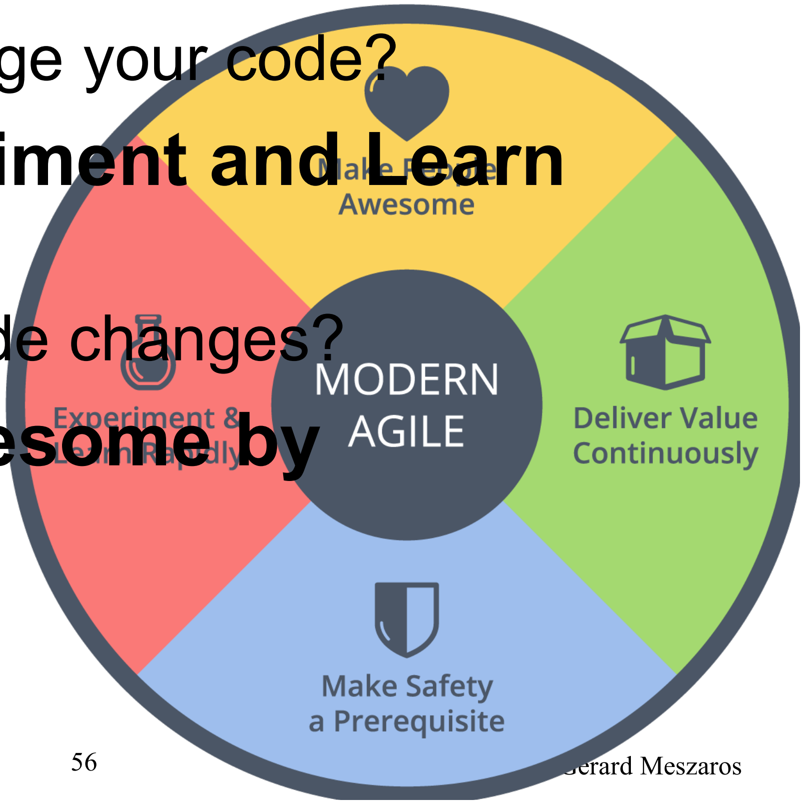
+ a bunch of other things …

+ Fanatical Attention to Test Maintainability

Robust Automated Tests

Closing Thoughts

- **Are your automated checks helping you deliver value continuously?**
  - Do they help you understand what you need to deliver?
- **Are your checks helping Make Safety a Prerequiste?**
  - Are they making it safer to change your code?
- **Are they helping you Experiment and Learn Continuously?**
  - Fast feedback on impacts of code changes?
- **Are you Making People Awesome by automating the checks?**
  - Happy developers and users?


Make People Awesome
MODERN AGILE
Experiment & Learn Rapidly
Deliver Value Continuously
Make Safety a Prerequisite

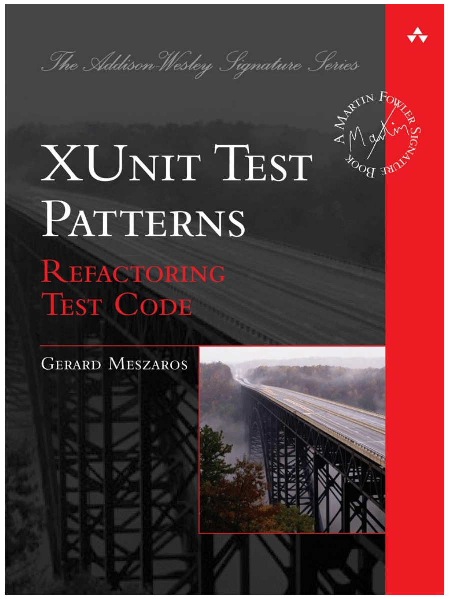
Thank You!

Gerard Meszaros

singapore2016@gerardm.com

http://www.xunitpatterns.com

Slides: http://singapore2016.xunitpatterns.com

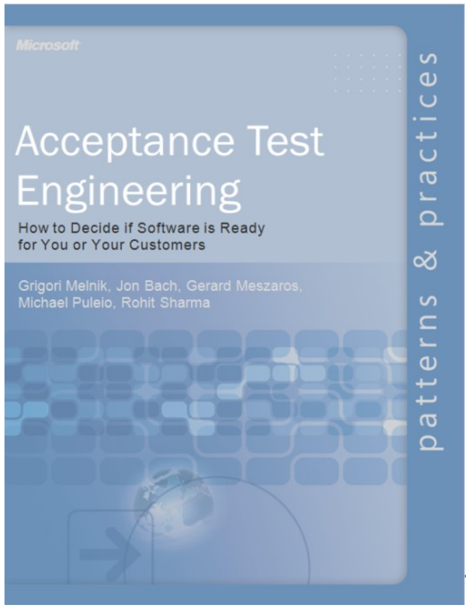

The Addison-Wesley Signature Series
XUnit Test Patterns
Refactoring Test Code
Gerard Meszaros
A Martin Fowler Signature Book

Jolt Productivity Award winner - Technical Books

Call me when you:

- **Want to transition to Agile or Lean**
- **Want to do Agile or Lean better**
- **Want to teach developers how to test**
- **Need help with test automation strategy**
- **Want to improve your test automation**

Available on MSDN:


Microsoft
Acceptance Test Engineering
How to Decide if Software is Ready for You or Your Customers
Grigori Melnik, Jon Bach, Gerard Meszaros, Michael Puleio, Rohit Sharma
patterns & practices