Maximizing Expressiveness of Automated Tests

Gerard Meszaros

Independent Consultant

CTO of FeedXL.Com

craft2014@gerardm.com

These Slides:
http://craft2014.testAutomationPatterns.com

My Background

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

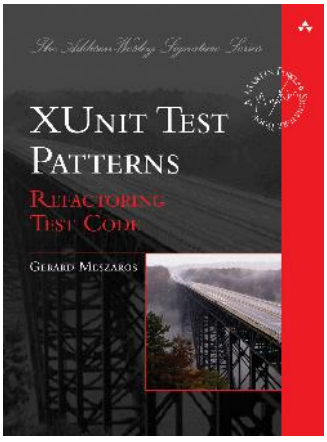
Embedded Telecom

- 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.

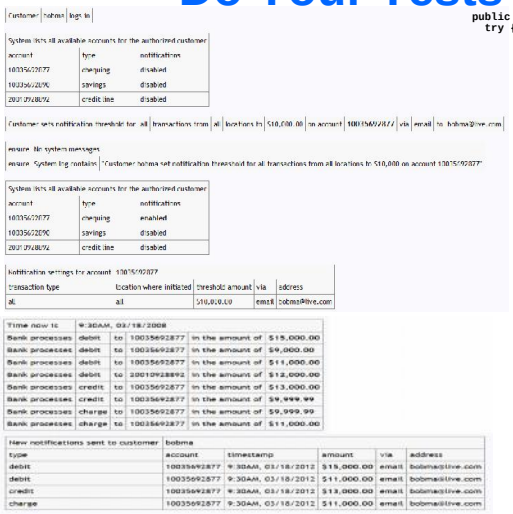


Gerard Meszaros
craft2014@gerardm.com

Agenda

- **Motivation**
  - Why Test Craftsmanship Matters
- **Managing Scope vs Detail**
  - In Functional (Acceptance) Tests
- **Achieving a Test-Friendly Architecture**
- **Managing Scope vs Detail**
  - In Unit Tests

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);
```

You might be questioning their value!

Where Automation Provides Value

- **Detecting Differences**
  - Find unexpected differences between different environments (e.g. different browsers)
- **"Pin the functionality"**
  - Find unexpected changes between versions as the product evolves over time
- **Avoid Wasting Effort**
  - testing manually when known issues exist
  - Testing things manually when it could be automated
  - Focus manual test effort on what automation doesn't

Where Automation Provides Value (2)

- **Communicating Expectations**
  - Avoid building the wrong product
  - Behaviour / Example-Driven Development (BDD / EDD)
  - Acceptance Test Driven Development (ATDD)
- **Find Issues Sooner**
  - By running the tests more frequently (ideally after every change)
  - Enabled by full automation (near zero cost)
  - Requires robust test automation

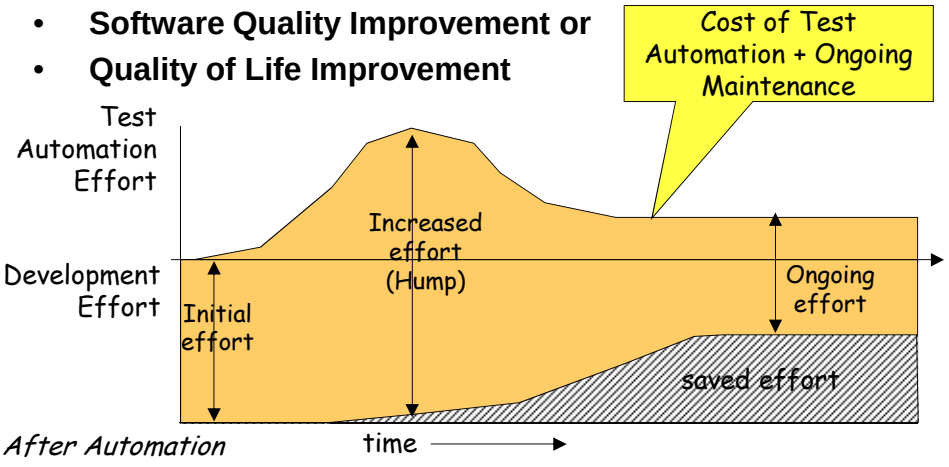
Where Automation Adds Cost

- **Direct Costs:**
  - Cost to write tests initially
  - Cost to investigate failing tests
  - Cost to fix failing tests
- **Exacerbated by:**
  - Too Many Tests
  - Too Fragile Tests
  - Too Complex Tests

Economics of Maintainability

Test Automation is a lot easier to sell on

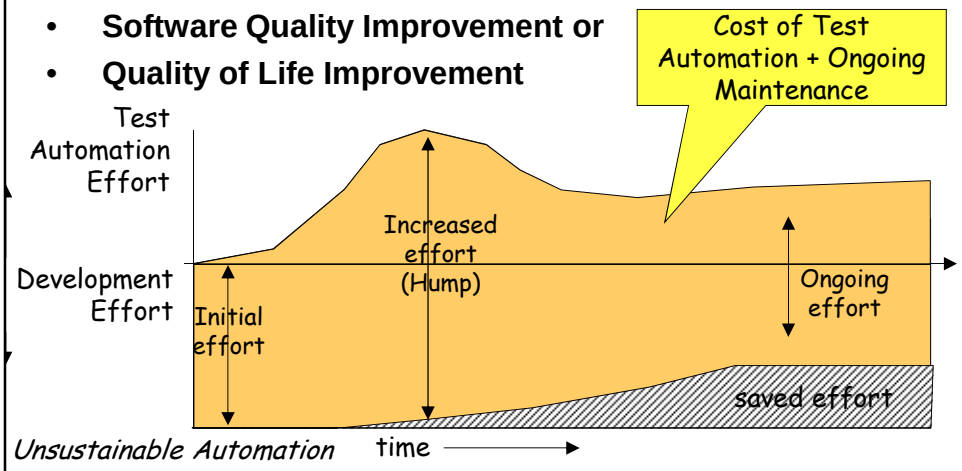
- **Cost reduction than**
- **Software Quality Improvement or**
- **Quality of Life Improvement**



Economics of Maintainability

Test Automation is a lot easier to sell on

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

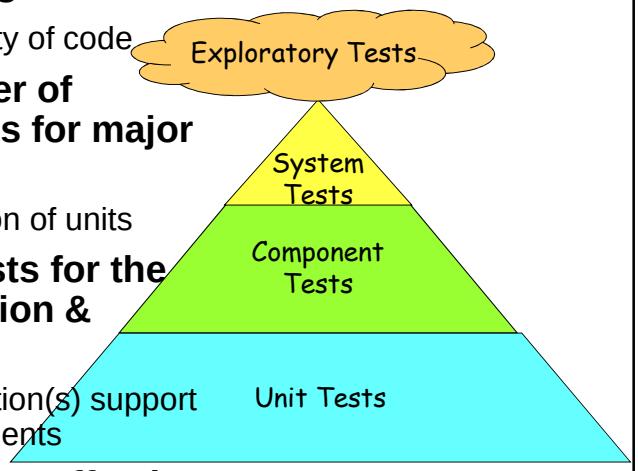


Agenda

- **Motivation**
- **Managing Scope vs Detail**
  - In Functional (Acceptance) Tests
- **Achieving a Test-Friendly Architecture**
- **Managing Scope vs Detail**
  - In Unit Tests

Test Automation Pyramid

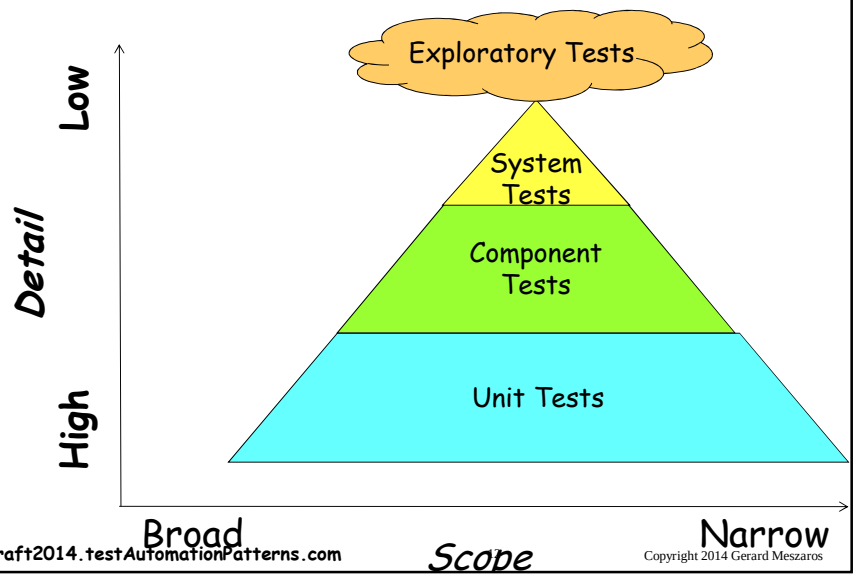
- **Large numbers of very small unit tests**
  - Ensures integrity of code
- **Smaller number of functional tests for major components**
  - Verify integration of units
- **Even fewer tests for the entire application & workflow**
  - Ensure application(s) support users' requirements
- **Tools to support effective exploratory testing**



http://craft2014.testAutom Pyramid originally proposed by Mike Cohn

Behavior Specification at Right Level

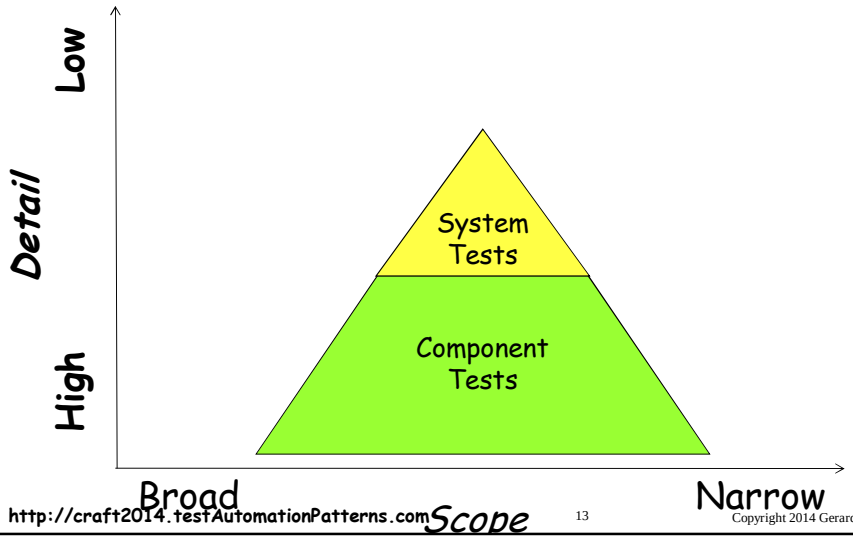
Need to specify at the right detail and scope.



http://craft2014.testAutomationPatterns.com

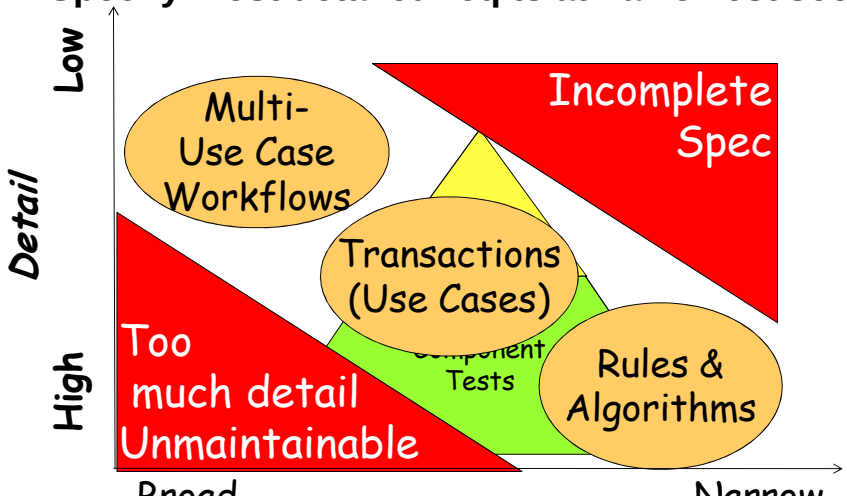
Behavior Specification at Right Level

Need to specify at the right detail and scope.

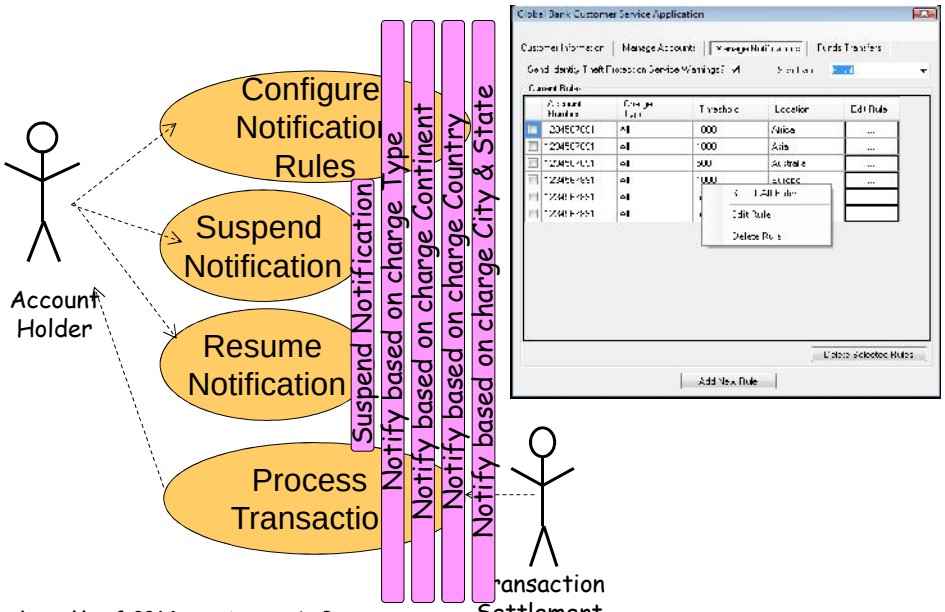


Behavior Specification at Right Level

- **Specify broad scope at minimum detail**
- **Specify most detailed req'ts at narrowest scope**



Use Cases & User Stories



Example:

Testing Notifications - 1

Customer	bobma	logs in

System lists all available accounts for the authorized customer		
account	type	notifications
10035692877	chequing	disabled
10035692890	savings	disabled
20010928892	credit line	disabled

Given: User and Accounts

When: Notification Rule is Configured

Customer sets notification threshold for	all	transactions from	all	locations to	$10,000.00	on account	10035692877	via	email	to	bobma@live.com

ensure	No system messages
ensure	System log contains	"Customer bobma set notification threashold for all transactions from all locations to $10,000 on account 10035692877"

System lists all available accounts for the authorized customer		
account	type	notifications
10035692877	chequing	enabled
10035692890	savings	disabled
20010928892	credit line	disabled

Notification settings for account	10035692877			
transaction type	location where initiated	threshold amount	via	address
all	all	$10,000.00	email	bobma@live.com

Then: Notification Rule is Active

Example:

Testing Notifications - 2

Use Case: Process Transactions

Time now is	9:30AM, 03/18/2008				
Bank processes	debit	to	10035692877	in the amount of	$15,000.00
Bank processes	debit	to	10035692877	in the amount of	$9,000.00
Bank processes	debit	to	10035692877	in the amount of	$11,000.00
Bank processes	debit	to	20010928892	in the amount of	$12,000.00
Bank processes	credit	to	10035692877	in the amount of	$13,000.00
Bank processes	credit	to	10035692877	in the amount of	$9,999.99
Bank processes	charge	to	10035692877	in the amount of	$9,999.99
Bank processes	charge	to	10035692877	in the amount of	$11,000.00

When: The Transactions to be processed

Then: Expected Notifications

New notifications sent to customer	bobma				
type	account	timestamp	amount	via	address
debit	10035692877	9:30AM, 03/18/2012	$15,000.00	email	bobma@live.com
debit	10035692877	9:30AM, 03/18/2012	$11,000.00	email	bobma@live.com
credit	10035692877	9:30AM, 03/18/2012	$13,000.00	email	bobma@live.com
			11,000.00	email	bobma@live.com

Broad Scope *(Multi-Actor);*
Medium Detail (*Too much for the scope)*

Issues With This Test

- **Difficult to understand which TX's should notify**
  - because cause (rules) and effect (notification) are far apart
- **Only verifies one simple combination of rules**
  - We will require many more tests to test all the other combinations
  - Lots of repetition of workflow & data across test cases
- **Simplest workflow;**
  - More complex workflows will be even longer and harder to understand
- **Tests will take a long time to run**
  - Due to need to configure first, then run transaction processing

What Can We Do?

- **Make tests shorter**
  - Fewer steps
- **Make tests less detailed**
  - Omit unnecessary information
- **Focus tests on specific aspects of behaviour**
  - Test/Spec Algorithms & Rules directly
  - Test less functionality, but more thoroughly

Raising abstraction level can help with this

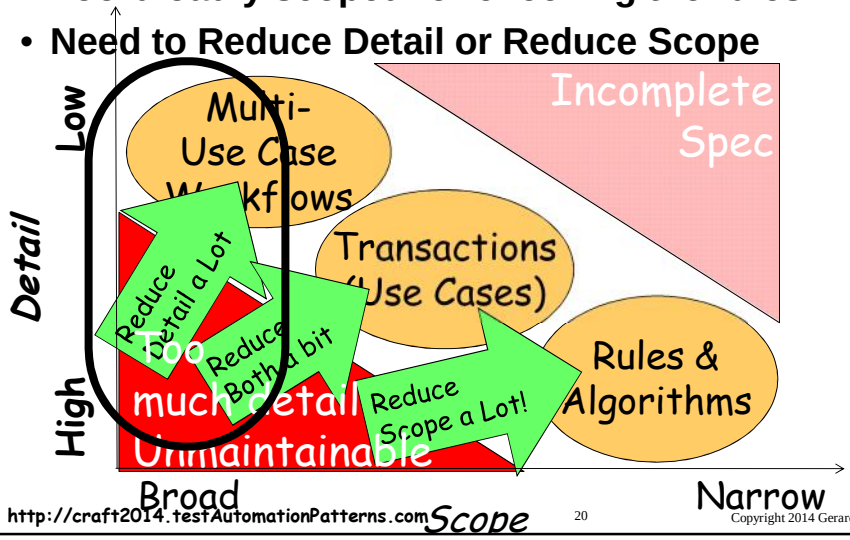
and with this

Requires Test-Friendly Architecture

Fine advice, but too vague

Changing Level of Abstraction/Detail

- **Too detailed for verifying workflow and**
- **Too broadly scoped for checking the rules**
- **Need to Reduce Detail or Reduce Scope**



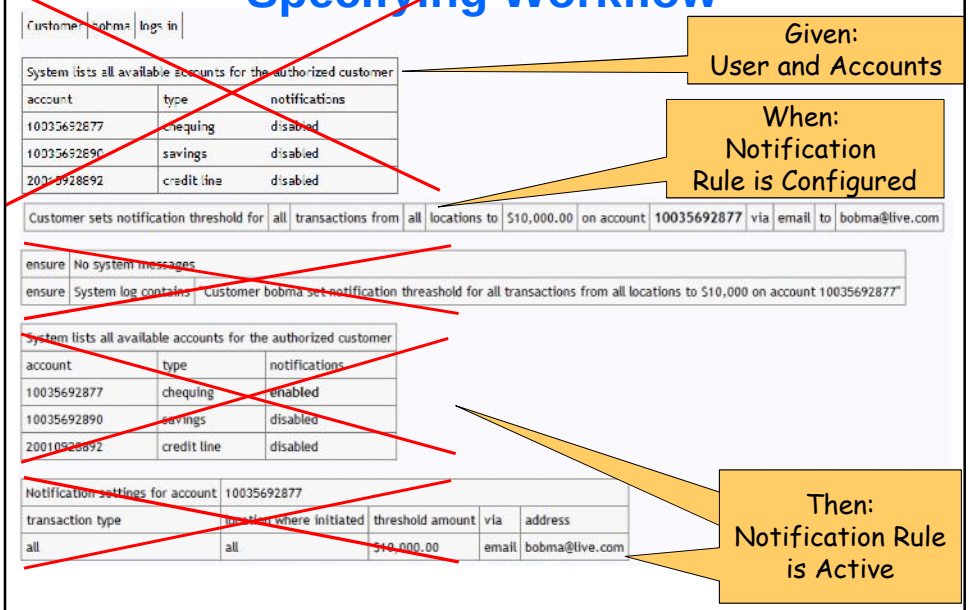
Omitting Unnecessary Information

- **Ask not what you can include in a test…**
- **Ask instead:**
- **What can I leave out of this test?**

"If it isn't essential to conveying the essence of the behavior, it is essential to <u>not</u> include it."

Example:

Specifying Workflow



Example:

Specifying Workflow

Customer sets notification threshold for	all	transactions from	all	locations to	$10,000.00	on account	10035692877	via	email	to	bobma@live.com

Example:

Specifying Workflow

Use Case: Process Transactions

Time now is	9:30AM, 03/18/2008				
Bank processes	debit	to	10035692877	in the amount of	$15,000.00
Bank processes	debit	to	10035692877	in the amount of	$9,000.00
Bank processes	debit	to	10035692877	in the amount of	$11,000.00
Bank processes	debit	to	20010928892	in the amount of	$12,000.00
Bank processes	credit	to	10035692877	in the amount of	$13,000.00
Bank processes	credit	to	10035692877	in the amount of	$9,999.99
Bank processes	charge	to	10035692877	in the amount of	$9,999.99
Bank processes	charge	to	10035692877	in the amount of	$11,000.00

When: The Transactions to be processed

Then: Expected Notifications

New notifications sent to customer	bobma				
type	account	timestamp	amount	via	address
debit	10035692877	9:30AM, 03/18/2012	$15,000.00	email	bobma@live.com
debit	10035692877	9:30AM, 03/18/2012	$11,000.00	email	bobma@live.com
credit	10035692877	9:30AM, 03/18/2012	$13,000.00	email	bobma@live.com
charge	10035692877	9:30AM, 03/18/2012	$11,000.00	email	bobma@live.com

Example:

Specifying Workflow

Time now is	9:30AM, 03/18/2008				
Bank processes	debit	to	10035692877	in the amount of	$15,000.00
Bank processes	debit	to	10035692877	in the amount of	$9,000.00
Bank processes	debit	to	10035692877	in the amount of	$11,000.00

New notifications sent to customer	bobma				
type	account	timestamp	amount	via	address
debit	10035692877	9:30AM, 03/18/2012	$15,000.00	email	bobma@live.com
debit	10035692877	9:30AM, 03/18/2012	$11,000.00	email	bobma@live.com

Example:

Specifying Workflow

Customer sets notification threshold for	all	transactions from	all	locations to	$10,000.00	on account	10035692877	via	email	to	bobma

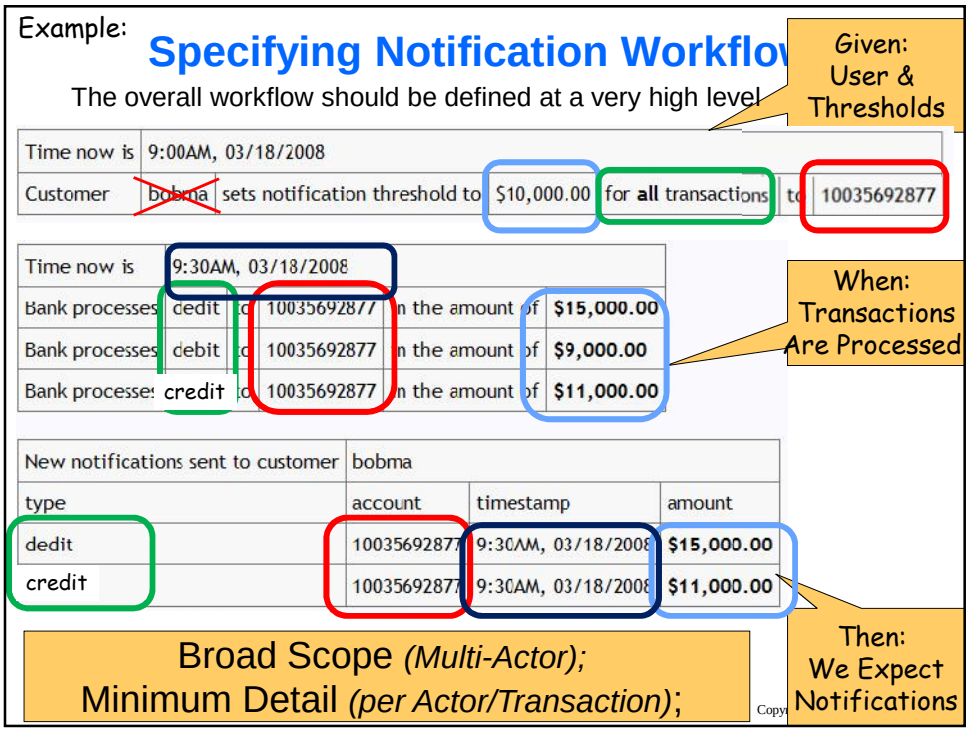
Time now is	9:30AM, 03/18/2008				
Bank processes	debit	to	10035692877	in the amount of	$15,000.00
Bank processes	debit	to	10035692877	in the amount of	$9,000.00
Bank processes	debit	to	10035692877	in the amount of	$11,000.00

New notifications sent to customer	bobma				
type	account	timestamp	amount	via	address
debit	10035692877	9:30AM, 03/18/2012	$15,000.00	email	bobma@live.com
debit	10035692877	9:30AM, 03/18/2012	$11,000.00	email	bobma@live.com

Example:

Specifying Notification Workflow

The overall workflow should be defined at a very high level



Broad Scope *(Multi-Actor);*
Minimum Detail *(per Actor/Transaction)*;

Specifying Suspension Workflow

Given BobMa has set notification threshold ~~to $10,000~~ for all transactions on his account

When the bank processes debit ~~for 15,000 to~~ his account (Over threshold)

Then BobMa receives notification ~~for debit 15,000~~

When BobMa suspends notification

And the bank processes debit ~~for 15,000 to~~ his account (Over threshold)

Then BobMa receives no notification

When BobMa resumes notification

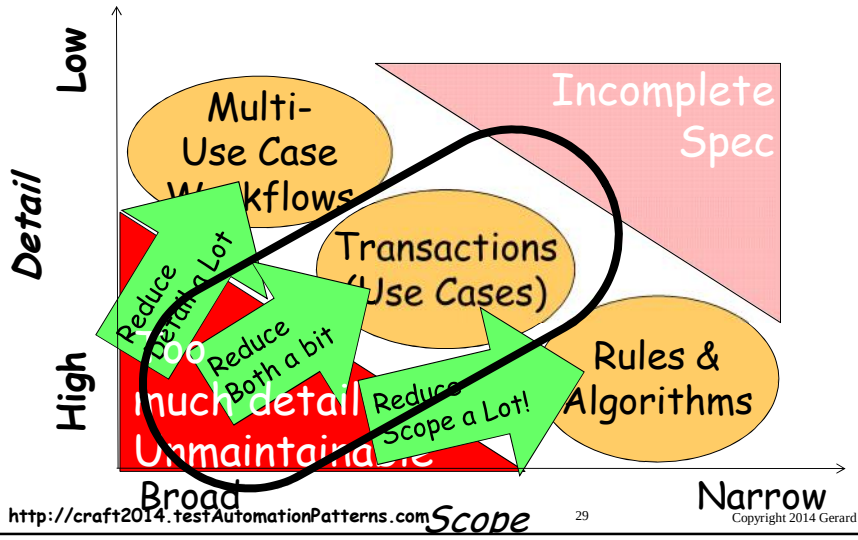
And the bank processes debit ~~for 15,000 to~~ his account (Over threshold)

Then BobMa receives notification ~~for debit 15,000~~

Broad Scope; Minimum Detail;
Irrelevant Details Omitted!

Changing Level of Abstraction/Detail

- So, where should omitted details go?



http://craft2014.testAutomationPatterns.com

Example:

Single Use Case Test

Use Case: Manage Notifications

Customer | bobma | logs in

System lists all available accounts for the authorized customer

account	type	notifications
10035692877	chequing	disabled
10035692890	savings	disabled
20010928892	credit line	disabled

Customer sets notification threshold for all | transactions from | all | locations to | $10,000.00 | on account | 10035692877 | via | email | to bobma@live.com

ensure No system messages

ensure System log contains | "Customer bobma set notification threshold for all transactions from all locations to $10,000 on account 10035692877"

Data to be shown on Manage Accounts Tab

System lists all available accounts for the authorized customer

account	type	notifications
10035692877	chequing	enabled
10035692890	savings	disabled
20010928892	credit line	disabled

Side effect of Adding A Notification

Notification settings for account 10035692877

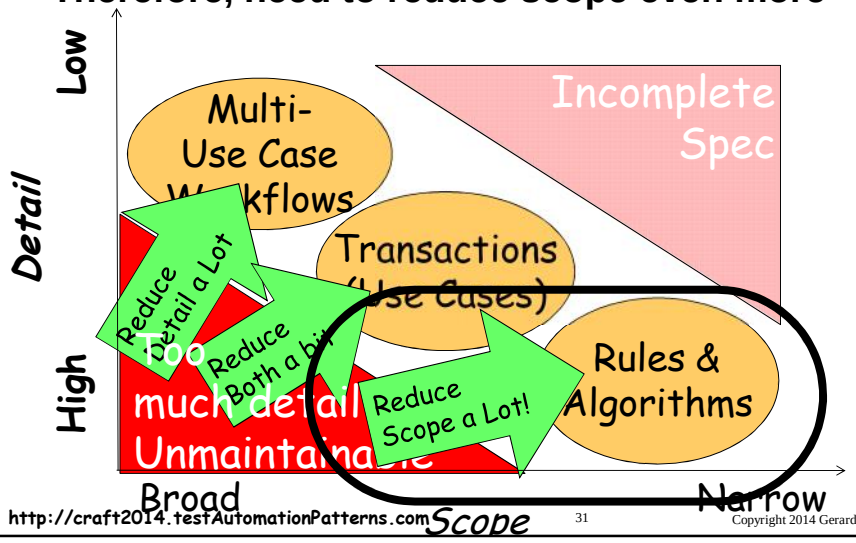
transaction type	location where initiated	threshold amount	via	address
all	all	$10,000.00	email	bobma@live.com

Data to be shown on Manage Notifications Tab

Medium Scope (Single Actor)
Medium Detail (Transaction, not UI);

Changing Level of Abstraction/Detail

- Need the detail to specify notification rules
- Therefore, need to reduce scope even more



Example:

Business Rule Specs

Threshold per Charge Type

When we ask NotificationRequired? with this transaction:

Configuration

Given these rules

CustomerAccounts[?]

Customer	Account	Label	Added()
bobma	100372	Checking	

CustomerThresholds[?]

Customer	Account	Charge Type	Threshold	Added()
bobma	100372	ALL	10,000	OK
bobma	100372	Travel	1,000	OK
bobma	100372	Restaurant	100	OK
bobma	100372	Groceries	264.23	OK

Process Transaction

NotificationRequired[?]

Account	Charge Type	Amount	Notify?
100372	Travel	999.99	No
100372	Travel	1,000.00	Yes
100372	Restaurant	99.99	No
100372	Restaurant	100.00	Yes
100372	Groceries	264.22	No
100372	Groceries	264.23	Yes
100372	Other	9.999.99	No
100372	Other	10,000.00	Yes

Narrow Scope (Single Rule)
High Detail (Everything that matters)

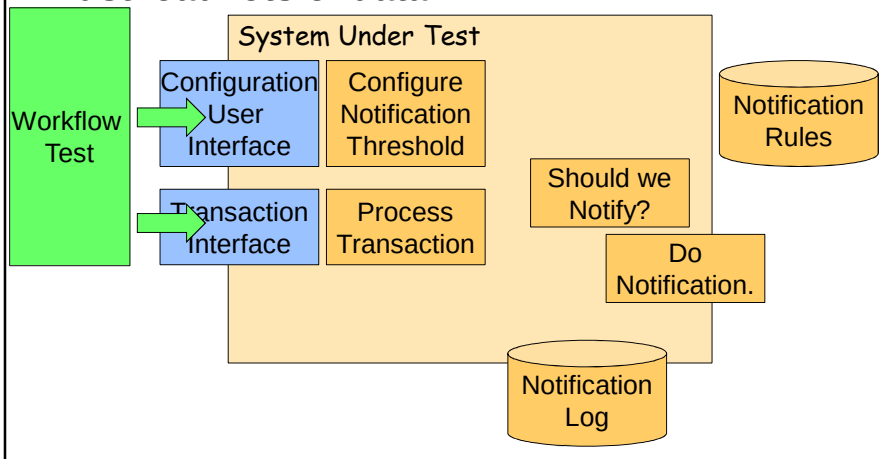
Then: The answer should be

Agenda

- **Motivation**
- **Managing Scope vs Detail**
  - In Functional (Acceptance) Tests
- **Achieving a Test-Friendly Architecture**
- **Managing Scope vs Detail**
  - In Unit Tests

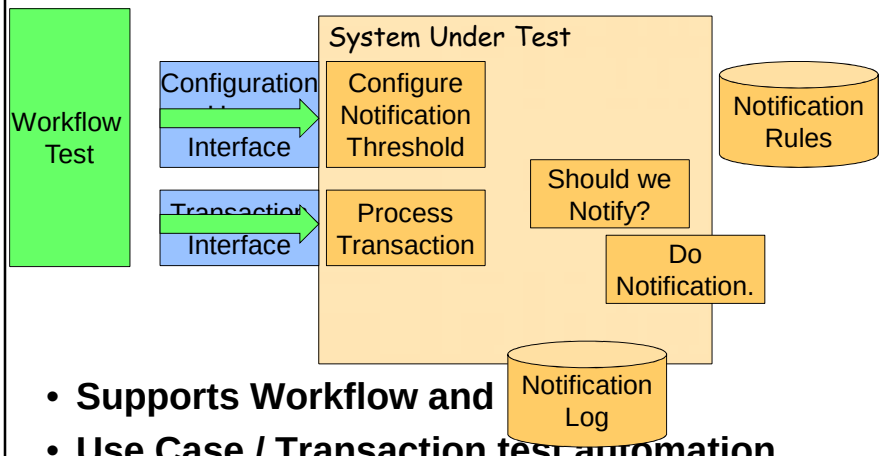
Test - After Architecture

- **Must test through User Interface**
- **Must load lots of data**



Test-Driven Architecture

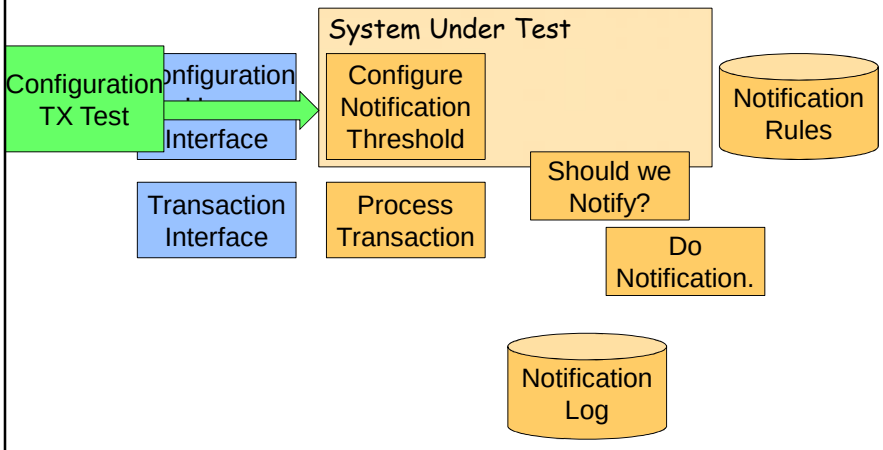
- **Need to provide API's to invoke functionality directly bypassing the UI**



- **Supports Workflow and**
- **Use Case / Transaction test automation**

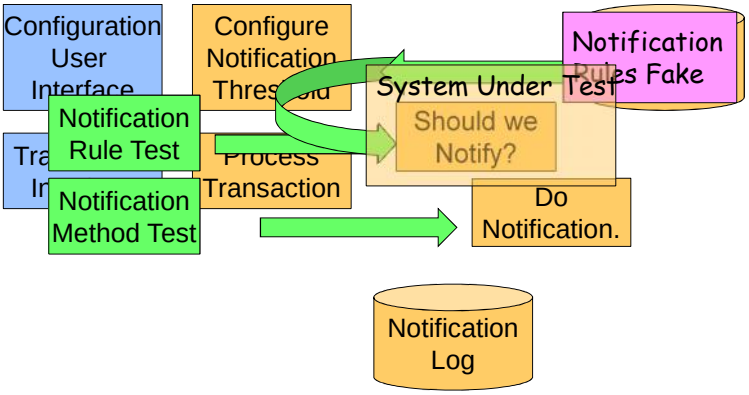
Test-Driven Architecture – Use Case Tests

- **Same API supports Transaction (Use Case) tests:**



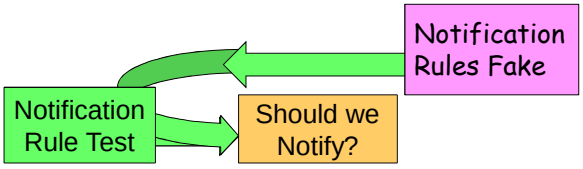
Test-Driven Architecture – Business Rules

- **Encapsulation of Rules in "Rules Component"**
  - which can be accessed directly via the Test



Business Rules Tests Encourage:

- **Encapsulation of Rules in "Rules Component"**
  - which can be accessed directly via the Test



- **Component Accept Rules via "Data Injection"**
  - Simplify Test Automation by avoiding need to configure
- **Forces Design-for-testability**
  - Only When Automation a Dev't Responsibility

Agenda

- **Motivation**
- **Managing Scope vs Detail**
  - In Functional (Acceptance) Tests
- **Achieving a Test-Friendly Architecture**
- **Managing Scope vs Detail**
  - In Unit Tests

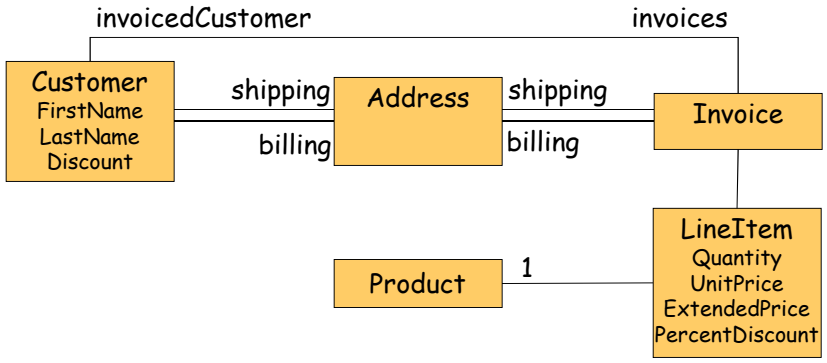
(Non-)Maintainable Test Code

```
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, "So
          new BigDecimal("19.99"));
    Invoice invoice = new Invoice(custome
    // 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());
```

```
public void
  testAddItemQuantity_severalQua
{ // Given:
  QUANTITY = 5 ;
  product = givenAnyProduct();
  invoice = givenAnEmptyInvoic
  // When:
  invoice.addItemQuantity(
          product, QUANTITY);
  // Then:
  assertExactlyOneLineItem(
    invoice,
    expectedItem(
          invoice,
          product, QUANTITY,
          product.getPrice()*
                QUANTITY) );
}
```

Example

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



The Whole Test

Given: ???

When we call addItemQuantity

Then: ???

```
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 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);
    }                    :
}
```

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

Refactoring

```
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");
}}
```

Refactoring

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

Pattern

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");
```

Pattern

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

Refactoring

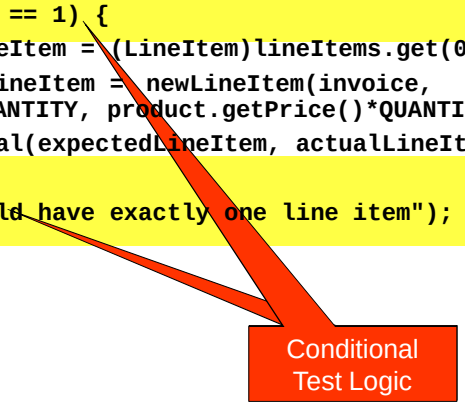
Introduce Custom Assert

```
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");
}
```

Custom Assertion

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");
}
```



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);
}
```


Hard-Coded Test Data

Code Smell

```
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

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 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(), getUniqueString(),
        getUniqueString(), getUniqueDiscount(),
        billingAddress, shippingAddress);
    Product product = new Product(
        getUniqueInt(), getUniqueString(),
        getUniqueNumber());
    Invoice invoice = new Invoice(customer);
```

Irrelevant Information (Obscure Test)

Pattern

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);
```


Code Smell

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)

Refactoring

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)

Refactoring

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);
}
```

Refactoring

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

Refactoring

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() {
   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 );
}
```

Given an empty invoice

when I call addItemQuantity

The invoice will end up with exactly 1 lineItem on it.

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

The Whole Test – Done

```
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

The Whole Test – Done

```
@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

The Whole Test – Done

```
@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

The Whole Test – Done

```
@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

The Whole Test – Done

```
@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

The Whole Test – Done

```
@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)  );
}
```

Constantly Strive to Improve Readability

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

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… {..}
}
```

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… {..}
}
```


GGM53

Rapid Test Writing

```
@Test public void
addItem_duplicateProduct_singleItemHasSumOfQuantities() {
   final int QUANTITY = 1 ;
   final int QUANTITY2 = 2 ;
   Product product1 = givenAnyProduct();
   Invoice invoice = givenAnEmptyInvoice();
   // When
   invoice.addItemQuantity(product1, QUANTITY);
   invoice.addItemQuantity(product1, 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..().

GGM53 Redo using new naming conventions

Gerard Meszaros, 10/19/2012

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;
   final int QUANTITY2 = 2;
   Product product1 = givenAnyProduct();
   Product product2 = givenAnyProduct();
   Invoice invoice = givenAnEmptyInvoice();
   // Exercise
   invoice.addItemQuantity(product1, QUANTITY);
   invoice.addItemQuantity(product2, QUANTITY2);
   // Verify
   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

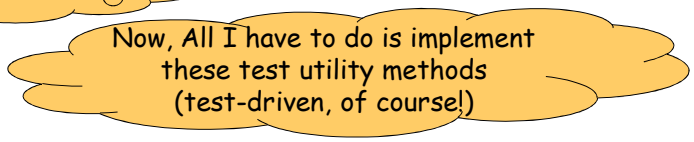
The invoice will end up with 2 lineItems on it, one for each of the two calls to add..().

Outside-In Development

```
@Test
public void GenerateInvoiceShould…() 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
```

Outside-In Development

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



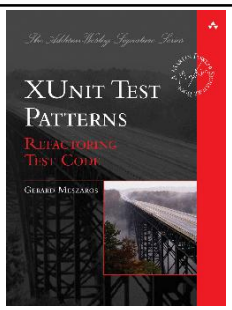
Closing Thoughts

- **Are you automating to find defects or prevent them?**
- **Are your automated tests good examples?**
  - Why not? What would you need to change?
- **Are your tests low maintenance?**
  - Why not? What causes them to break?
  - What could you change to make them break less often?
  - .... to reduce the impact of breakage?

Thank You!

Gerard Meszaros
craft2014@gerardm.com
http://www.xunitpatterns.com

Slides: http://craft2014.testAutomationPatterns.com



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: