improving
It's what we do. ™

Acceptance & Test-Driven Development

Improving Enterprises

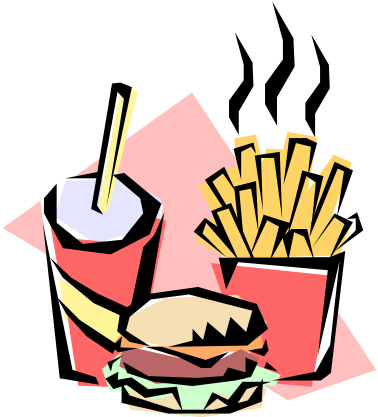
improving
It's what we do. ™

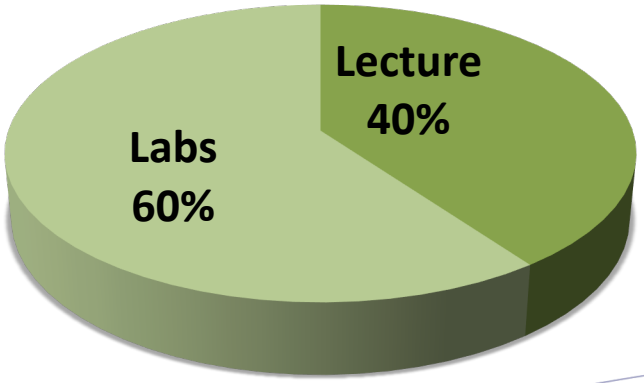
INTRODUCTIONS

Introductions

- Hello!
- About us
- About you
  - Name
  - Title/function
  - Experience
  - **Expectations for the class**

Logistics





Agenda

DAY 1 (What/Why)

- Unit Testing / TDD / BDD
- User Stories
- Acceptance Testing

DAY 2 (Doing)

- Specification by Example
- Pair Programming
- Test Doubles

DAY 3 (Keeping)

- Testing Legacy Code
- Maintaining Tests

improving
It's what we do. ™

Module 1:

EXTREME PROGRAMMING

Thoughts On These Companies?

How to work

- New world of software development
  - More change
  - Shorter decision cycles
- Every business is a technology business
- Fastest to market wins
  - Fast organizations beat slow organizations

Its Not Just About Code Anymore

NETFLIX

- 60 million tests
- 4000 deploys per day*

- Google reached that number of tests and daily deploys in 2012.**

Google

- 2 billion lines of code
- Accessible to 25000 developers
- Changes 15 million lines a week
- 1 Repository
- Developers do instant deploys to production
- All on Mainline

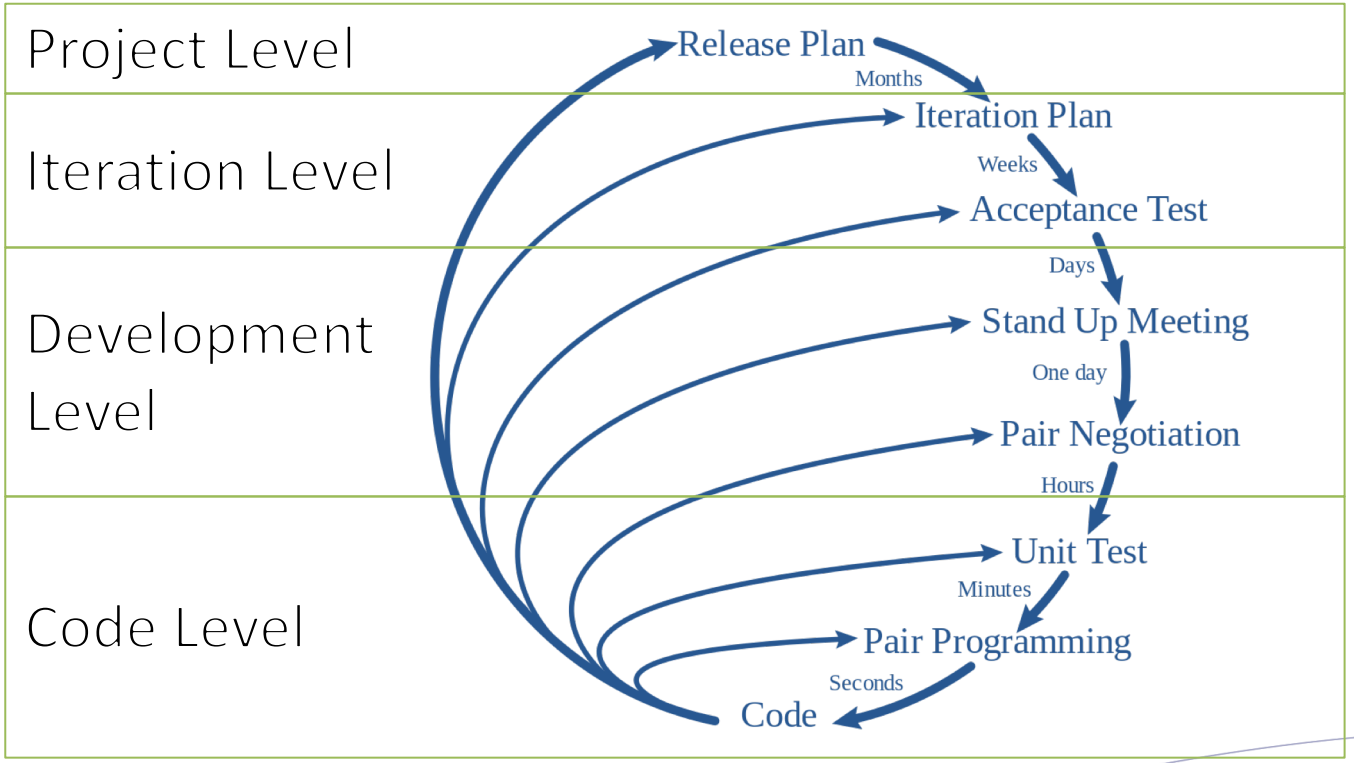
Test First is a Prerequisite

Continuous Deployment requires Continuous Integration which requires Test First Development

Extreme Programming (XP)

XP is about improving **software quality** and **responsiveness** to **changing customer requirements**.

XP Planning/Feedback Loops



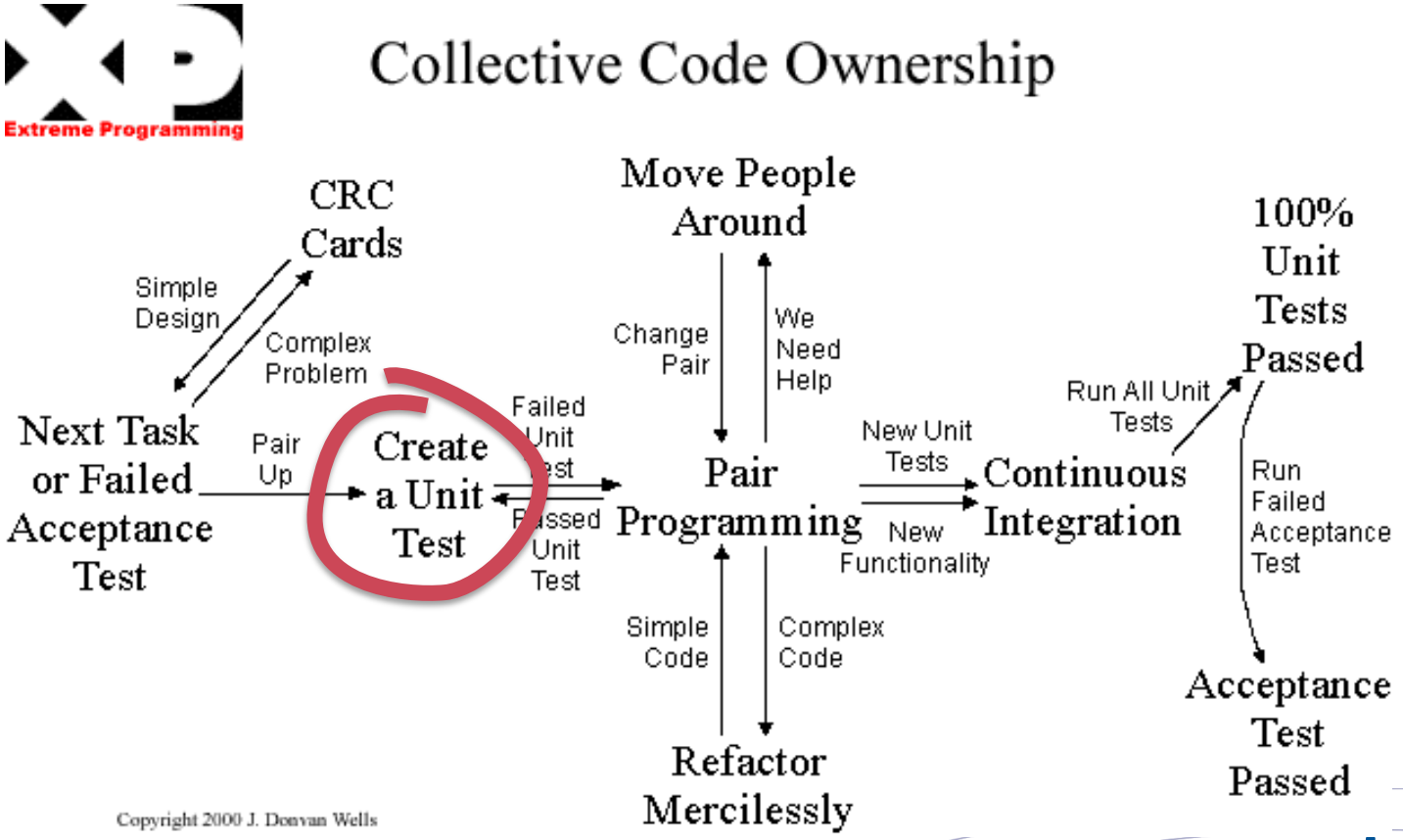
Module 2:

UNIT TESTING

You will learn to

- Understand the motivation behind unit testing
- Structure tests with four distinct phases:
  Setup, Exercise, Verify, Teardown
- Use a unit testing tool

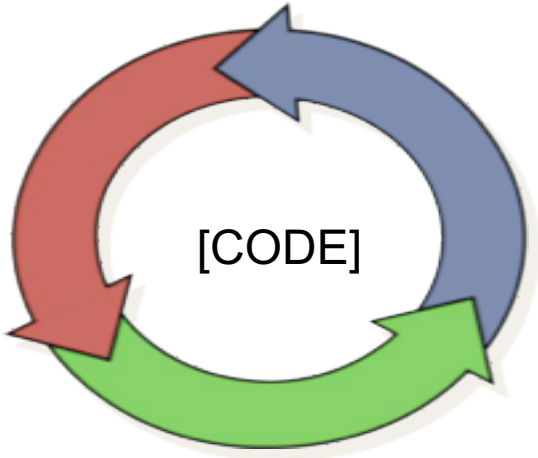
XP – Code Level



Familiar?

"I don't want to touch that! It'll take forever, and I don't know what else will break if I do. I'll just rewrite it."

Unit Testing



"Alarm System"

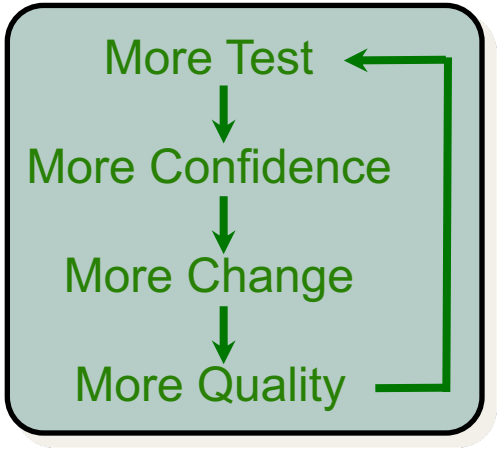
Regression testing

Reduce number of defects

Fix defects quicker

Confidence in the code

"Virtuous Cycle"



Unit Testing

What is a Unit?

Smallest testable part of an application

What is a Test?

Validation that the code is working properly

You have already done it...

...but it may not have been:

- ✓ Consistent
- ✓ Automated
- ✓ Shared
- ✓ As easy as it could be

Unit Testing - Traditional

C#

```
public class MyClassTest {

    [TestMethod]
    public void TestMethod1() {
        MyClass instance = new MyClass();

        instance.Method1();

        Assert.AreEqual("expected", instance.Value);
    }

    [TestMethod]
    public void TestMethod2() {
        MyClass instance = new MyClass();

        Result result = instance.Method2();

        Assert.IsNotNull(result);
        Assert.AreEqual("expected", result.Value);
    }

}
```

Setup

Exercise

Verify

Teardown

```
AreEqual(expected, actual);
AreNotEqual(expected, actual);
Greater(expected, actual);
Less(expected, actual);
IsFalse(condition);
IsTrue(condition);
IsNull(object);
IsNotNull(object);
IsEmpty(object);
AreSame(expected, actual);
AreNotSame(expected, actual);
Pass();
Fail();
```

Unit Testing - Traditional

Java

```
public class MyClassTest {

    @Test
    public void testMethod1() {
        MyClass instance = new MyClass();

        instance.method1();

        assertEquals("expected", instance.getValue());
    }

    @Test
    public void testMethod2() {
        MyClass instance = new MyClass();

        Result result = instance.method2();

        assertNotNull(result);
        assertEquals("expected", result.getValue());
    }

}
```

Setup

Exercise

Verify

Teardown

```
assertEquals(expected, actual);
assertFalse(condition);
assertTrue(condition);
assertNull(object);
assertNotNull(object);
assertSame(expected, actual);
assertNotSame(expected, actual);
fail();
```

Unit Test Example

Let’s try it...

improving
It's what we do. ™

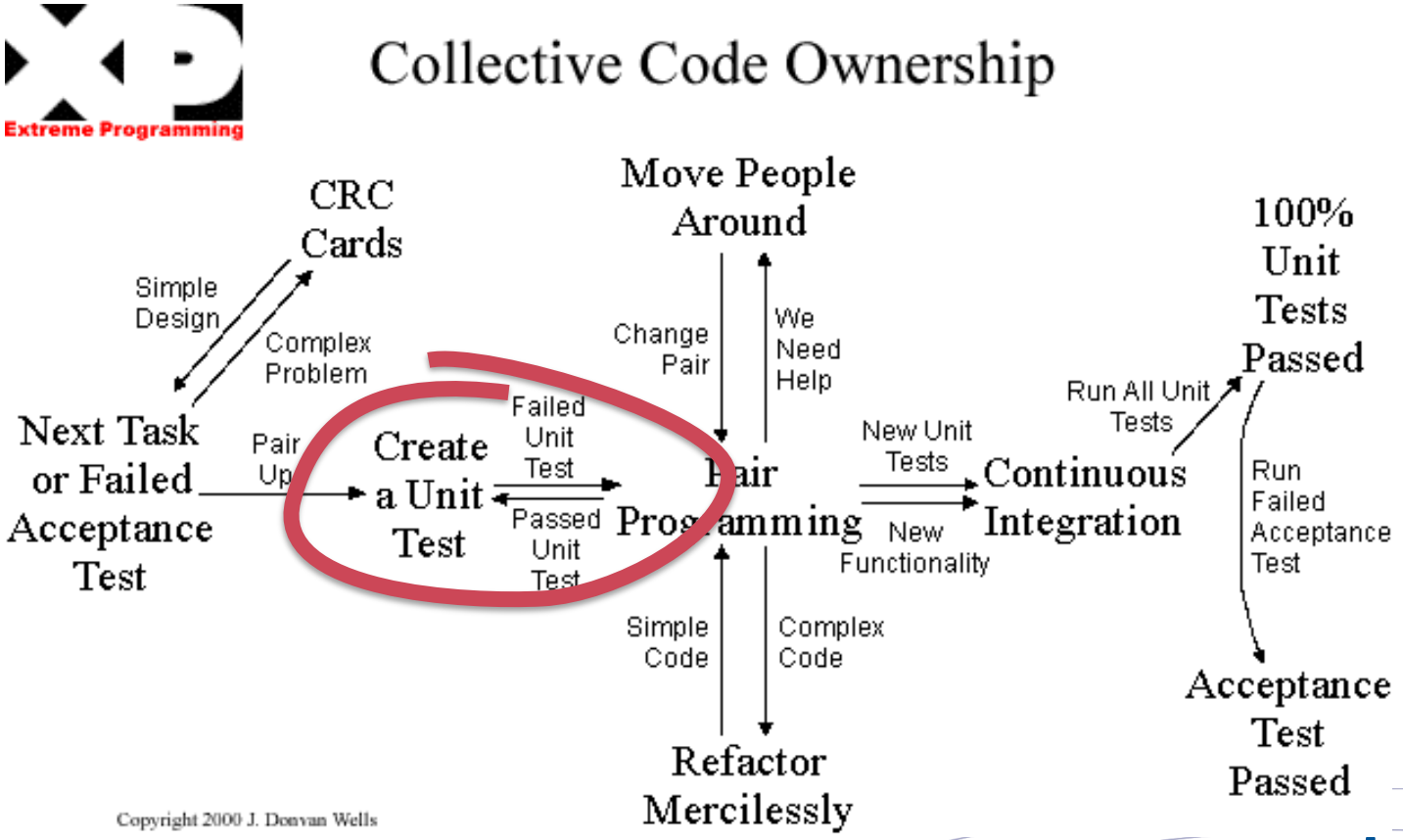
Module 3:

TEST-DRIVEN DEVELOPMENT

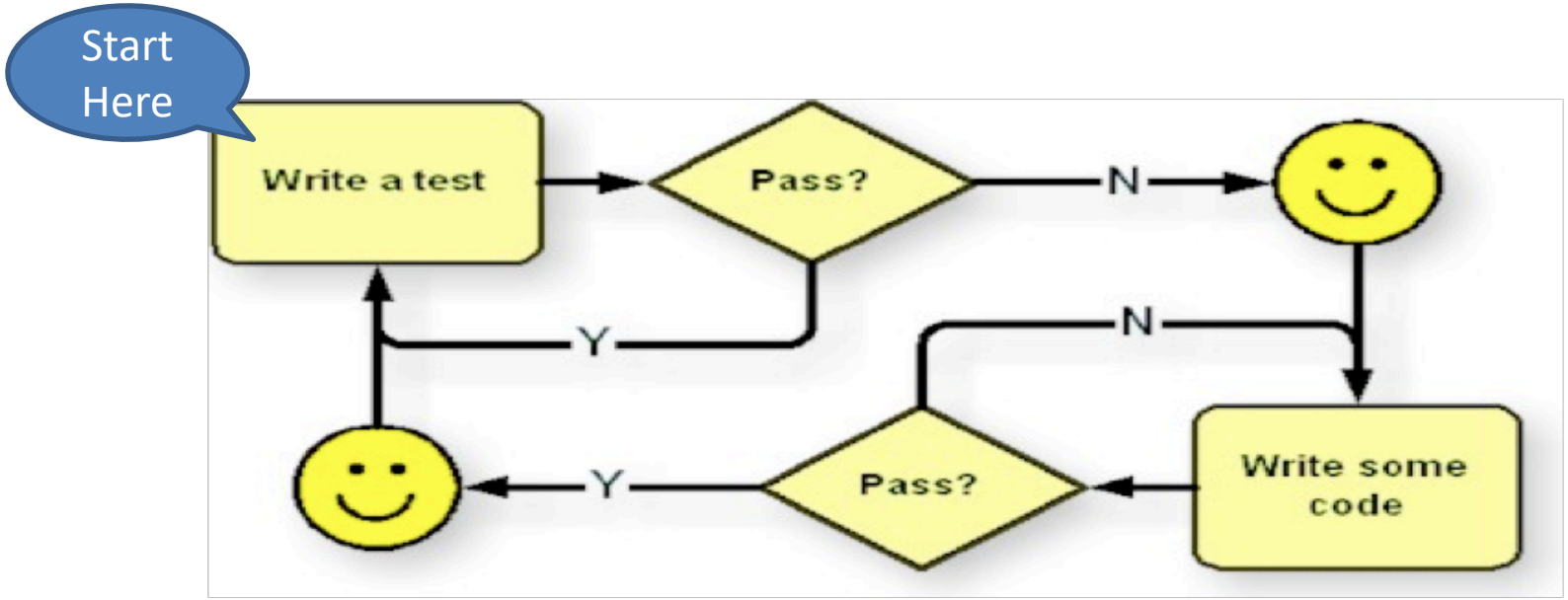
You will learn to

- Develop code using the TDD cycle
- Design good tests so that they are independent and fast
- Follow the TDD Mantra: Red, Green, Refactor
- Distinguish TDD from BDD (Behavior-Driven Development)

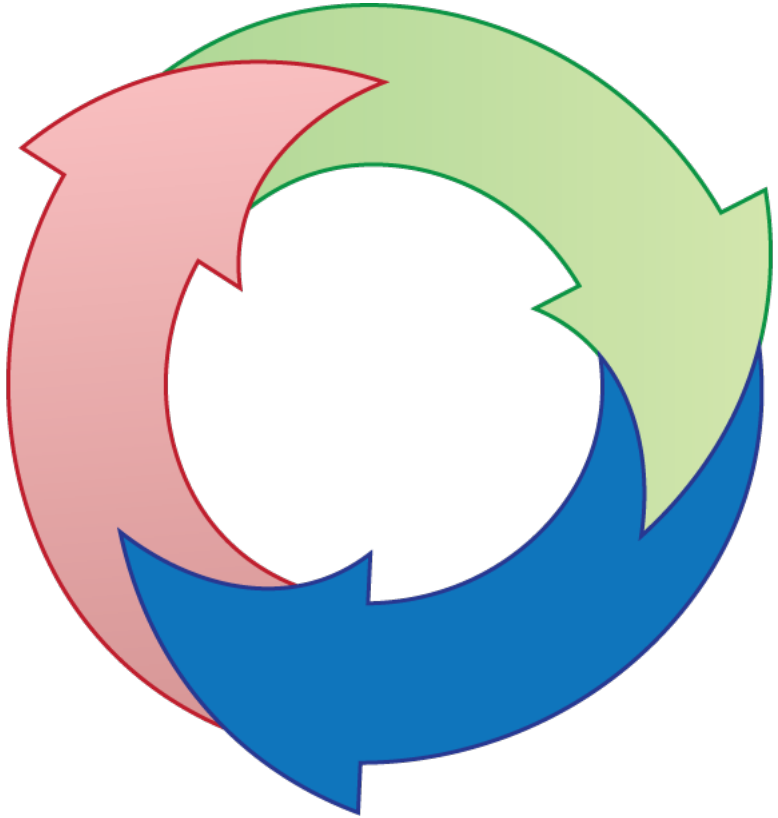
XP – Code Level



Test Driven Development



TDD Mantra



Test (Red)

Code (Green)

Design (Refactor)

Step 0: Think
(take as long as you need*)

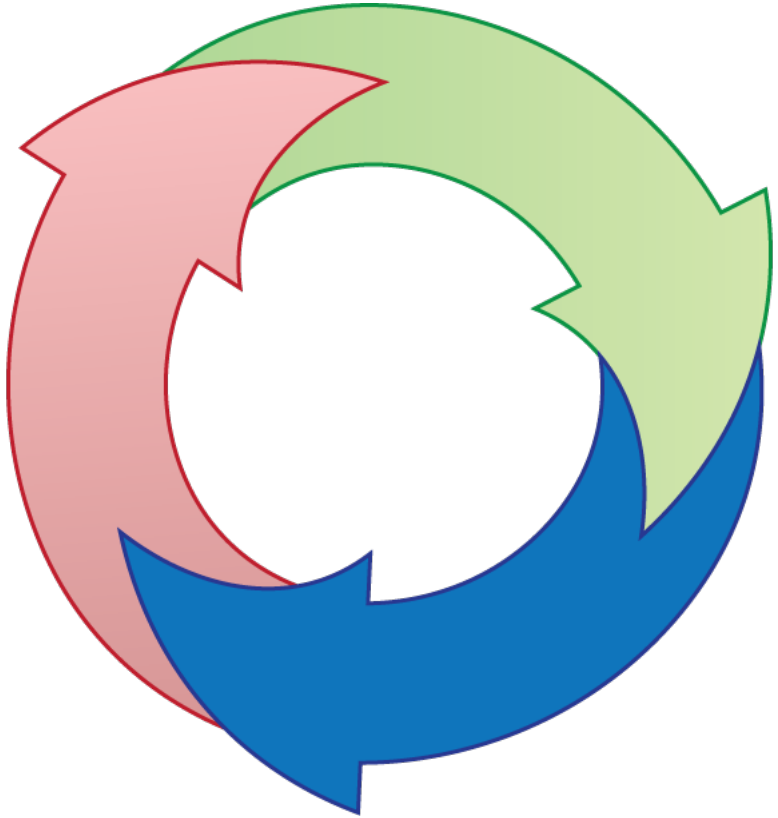
* - Jim Shore

Step 1: Write a Spec

Step 2: Make it Pass (just barely)

Step 3: Refactor

TDD Mantra



Test (Red)

Code (Green)

Design (Refactor)

2-10 minutes

TDD/BDD Lets You Know When…

Ya

Ain't

Gonna

Need

It

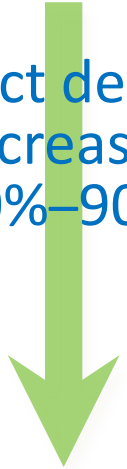
So…

“Only write code to fix a failing test”

Case Study

Empirical studies compared 4 projects, at Microsoft and IBM, that used TDD with similar projects that did not use TDD.

The findings were similar to other case studies:



Defect density decreased 40%–90%

Development time increased by 15–35%

(though teams agreed that this was offset by reduced maintenance costs)

http://www.infoq.com/news/2009/03/TDD-Improves-Quality

TDD Example

Let’s try it...

Unit Testing – BDD Style

```
public class MyClassBehavior {

    @Test
    public void shouldDoThisAndThat() {
        MyClass instance = new MyClass();

        instance.method1();

        assertEquals("expected", instance.getValue());
    }

    @Test
    public void shouldDoThatAndThis() {
        MyClass instance = new MyClass();

        Result result = instance.method2();

        assertNotNull(result);
        assertEquals("expected", result.getValue());
    }

}
```

```
assertEquals(expected, actual);
assertFalse(condition);
assertTrue(condition);
assertNull(object);
assertNotNull(object);
assertSame(expected, actual);
assertNotSame(expected, actual);
fail();
```

Behavior-Driven Development

Forget about **Tests**; Think about **Specifications**

- it's easy to come up with reasons not to test, but writing specs can become an integral part of development

Forget about **Units**; Think about **Behaviors**

- worry more about how a developer will interact with the code, and not what methods a class has

Example

```
public class ItemBehavior {
  @Test
  public void shouldCheckIfInventoryIsLow() {
    Item item = new Item();
    item.setQuantity(20);
    assertFalse(item.isLowInventory());
    item.setQuantity(2);
    assertTrue(item.isLowInventory());
  }
  @Test
  public void shouldStoreItemInformation() {
    Item item = new Item("iPhone", 400.00, 10);
    assertEquals("iPhone", item.getName());
    assertEquals(400.00, item.getPrice());
    assertEquals(10, item.getQuantity());
  }
}
```

Given an Item…

Item
name Price quantity
isLowInventory()

…what should generate XML for an item?

Example

Does this make sense?

```
public class ItemBehavior {
  @Test
  public void shouldCheckIfInventoryIsLow() {...}

  @Test public void shouldStoreItemInformation() {...}

  @Test
  public void shouldGenerateXmlForAnItem() {...}
 }
```

Example

How about a new spec?

```
public class XmlParserBehavior {
  @Test
  public void shouldGenerateXmlForAnItem() {
    Item item = new Item("iPhone", 400.00, 10);
    Element itemXml = XmlParser.generateXmlFor(item);
    assertNotNull(itemXml);
    // ... verify the xml ...
  }
}
```

What else “should” it do?

Example

```
public class XmlParserBehavior {
  @Test
  public void shouldGenerateXmlForAnItem() {...}

  @Test
  public void shouldCreateAnItemFromXml() {
    Element itemXml = // ... build xml ...
    Item item = XmlParser.createItemFrom(itemXml);
    // ... verify the item ...
  }

  @Test(expected=IllegalArgumentException.class)
  public void shouldFailXmlGenerationWhenNullObject() {
    Item item = null;
    Element itemXml = XmlParser.generateXmlFor(item);
    fail("Must throw IllegalArgumentException");
  }
}
```

So without actually creating the code for the class under test, we have effectively defined it's interface.

We are designing.

This is Test-Driven Design.

BDD Example

Let’s try it...

improving
It's what we do. ™

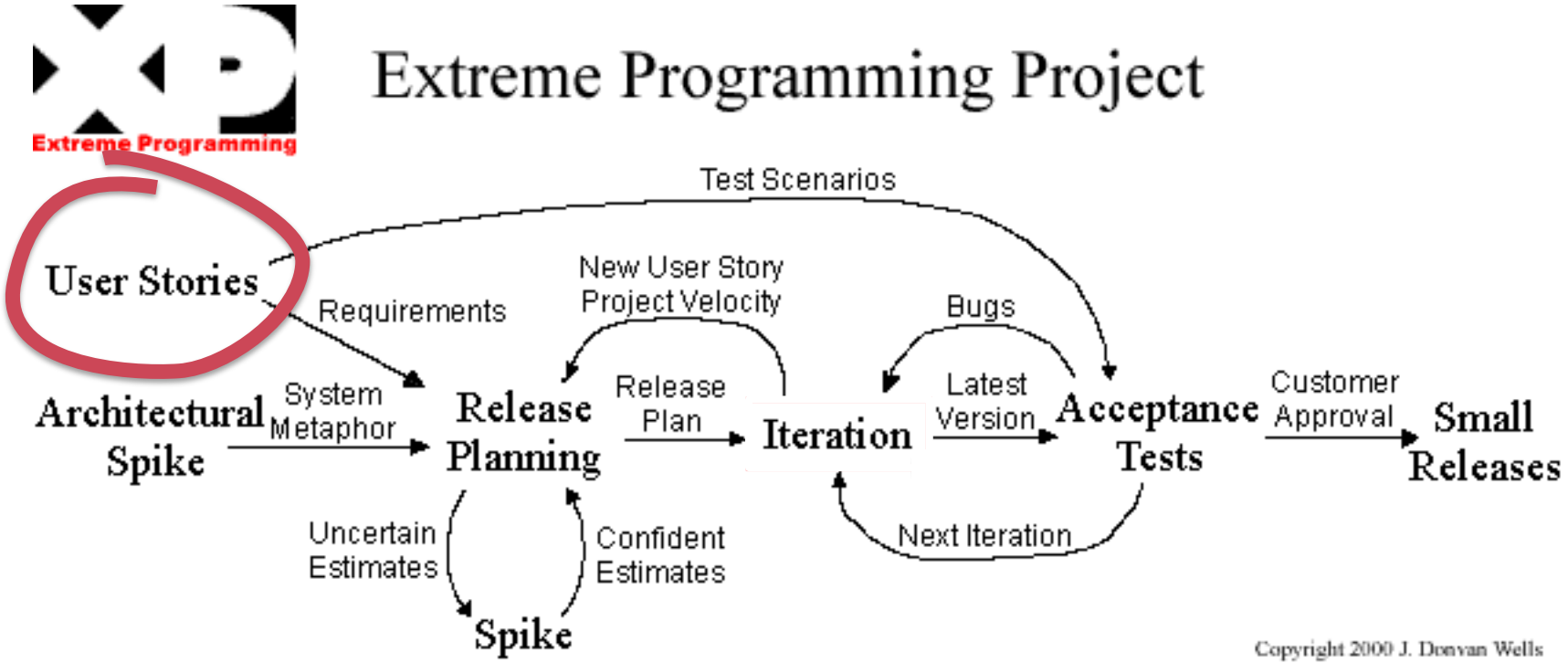
Module 4:

USER STORIES

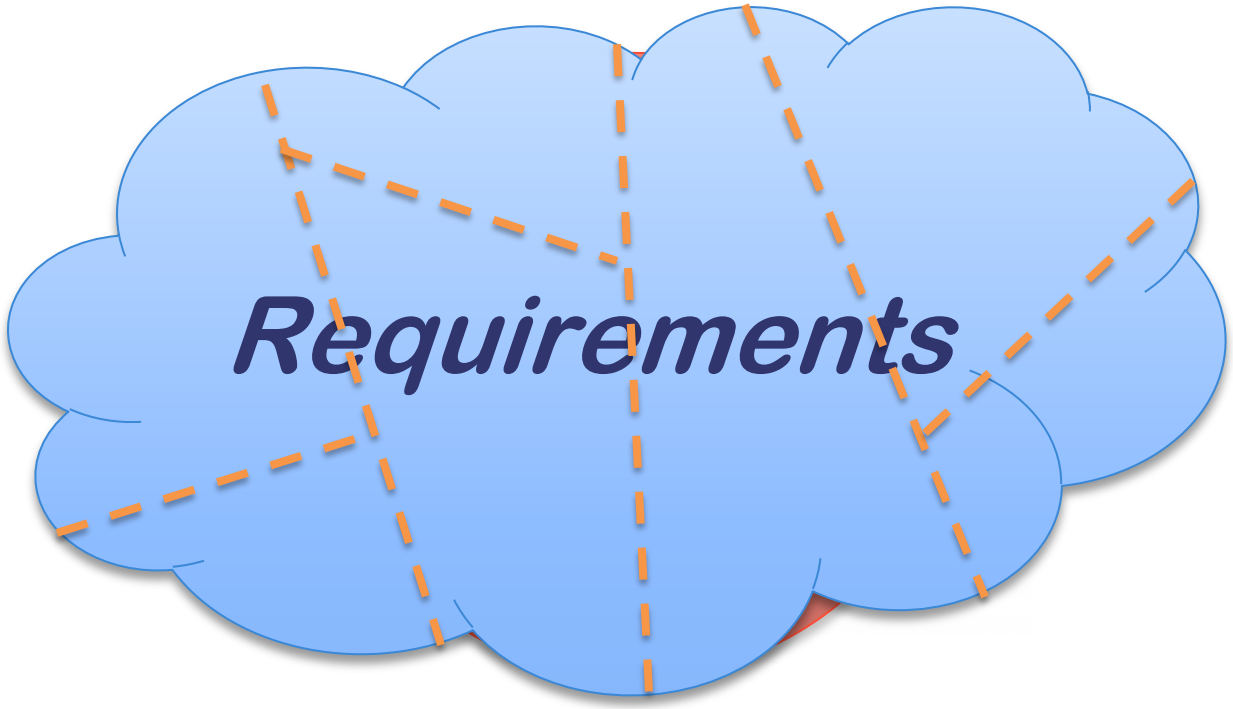
You will learn to

- Understand the motivation behind User Stories
- Distinguish between good and bad User Stories
- Recall the 3 Cs of User Stories
- Use a User Story template

XP - Project Level



What are Requirements?



What are Requirements?

What is your goal?

To Document?

- IEEE 830
- Detailed Use cases

To Represent?

- User Stories
- Use Case Briefs

The Big Picture

Getting Things Done.

Organize Thoughts by:

Creating a List

Prioritizing the List

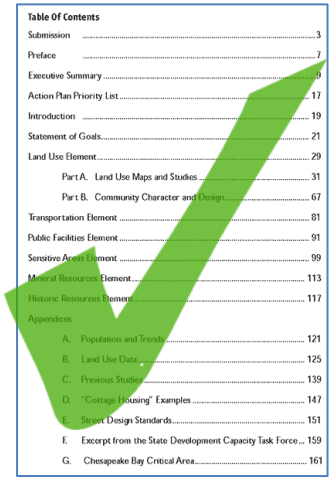
Using the List

User Stories

Inch deep / Mile wide (TOC, not a Book)

Represents the requirements Inventory

Done in ~ a day



Quiz!

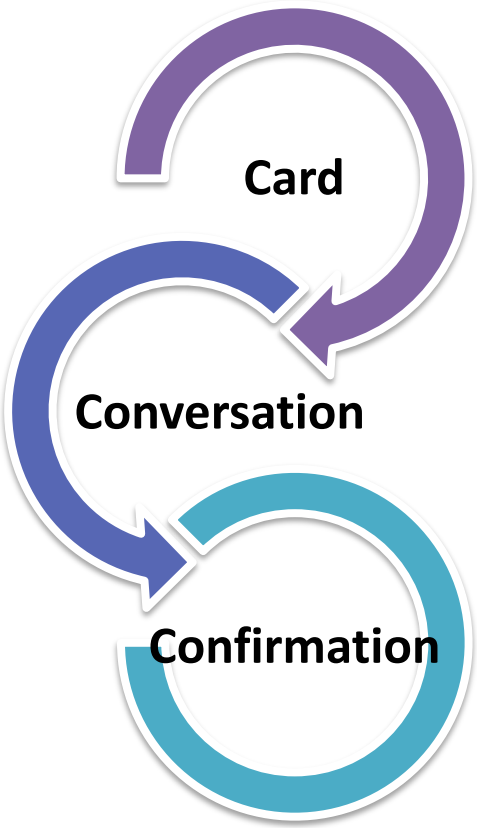
Are the following

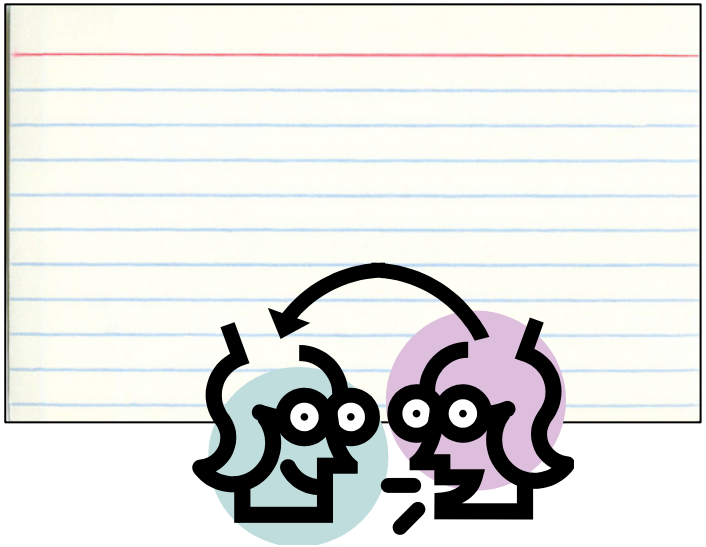
User Stories valid?

Valid Product Backlog Items

- Feature Requests
- User Stories
- Bugs / Defects
- Use Cases
- Experiments
- Non-Functional Requirements

The 3 C’s of User Stories





User Story Template

As a <stakeholder>,
I want to
<do something>
so that I can
<accomplish some goal>.

Who?

What?

Why?

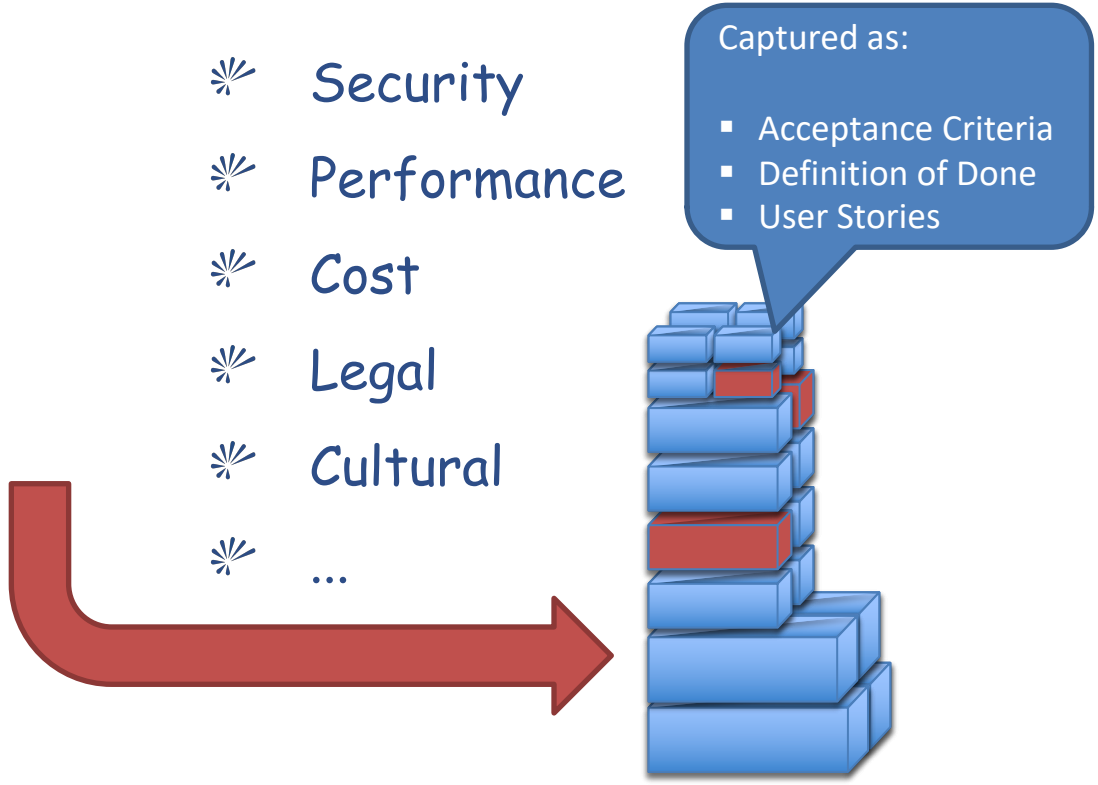
User Story Example

As a Blog Reader,
I want to
comment on a blog entry
so that I can
contribute to the conversation

Non-Functional Requirements

- Usability
- Scalability
- Portability
- Maintainability
- Availability
- Accessibility
- Supportability

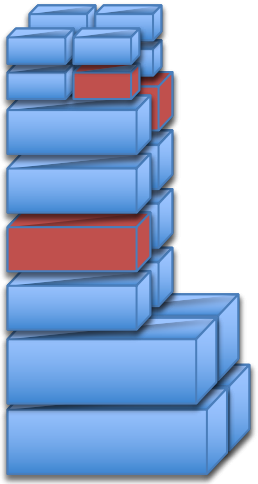
- Security
- Performance
- Cost
- Legal
- Cultural
- …



… but one is different

- Usability
- Scalability
- Portability
- **Maintainability**
- Availability
- Accessibility
- Supportability

- Security
- Performance
- Cost
- Legal
- Cultural
- …



Non-Functional Story Example

As a Blogger, I want to
support many readers
so that I can
reach a wider audience

Good Stories Are...

Independent

Negotiable

Valuable

Estimable

Small

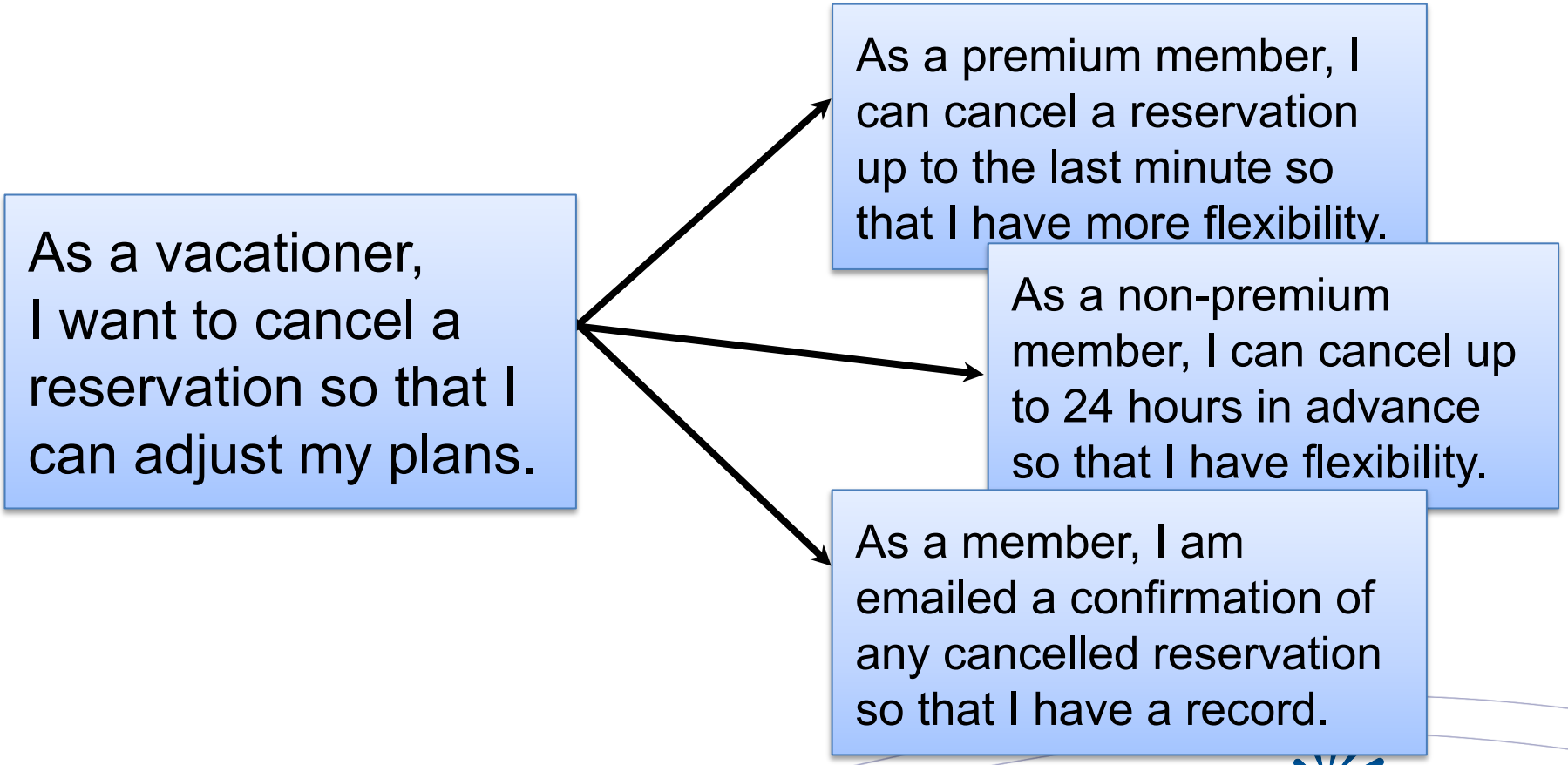
Testable

Confirmation is included as acceptance criteria

As a vacationer,
I want to cancel a reservation
so that I can adjust my plans.

- ❑ Verify that a premium member can cancel the same day without a fee.
- ❑ Verify that a non-premium member is charged 10% for a same-day cancellation.
- ❑ Verify that an email confirmation is sent.
- ❑ Verify that the hotel is notified of any cancellation.

Breaking Down Stories



So User Stories…

- are like to-do list items
- describe value from the business' perspective
- are not meant to be THE requirements
- can be divided by its acceptance criteria

improving
It's what we do. ™

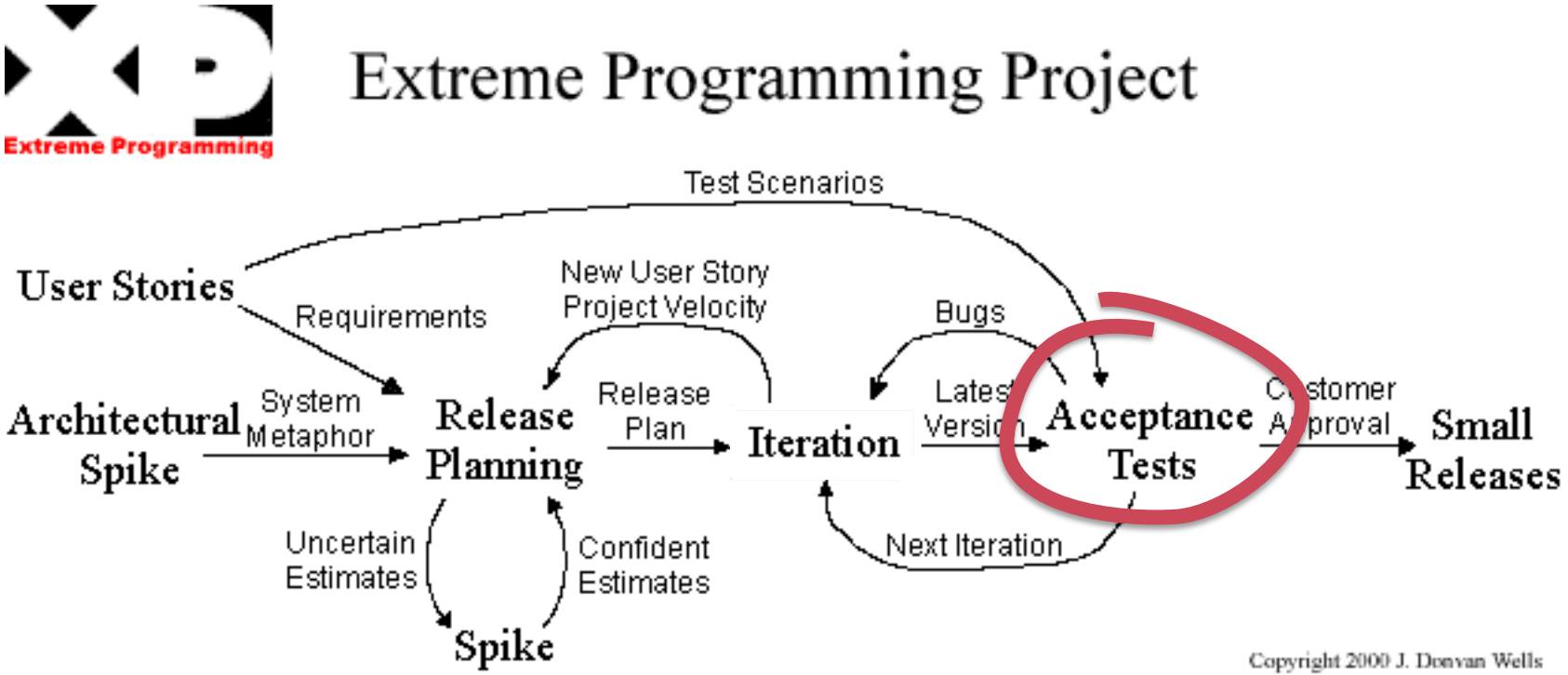
Module 5:

ACCEPTANCE TESTING

You will learn to

- Write tests for a User Story
- Distinguish between a test and a requirement
- Recognize who is responsible for Acceptance Tests
- Identify different types of tests
- Use different Acceptance Test templates

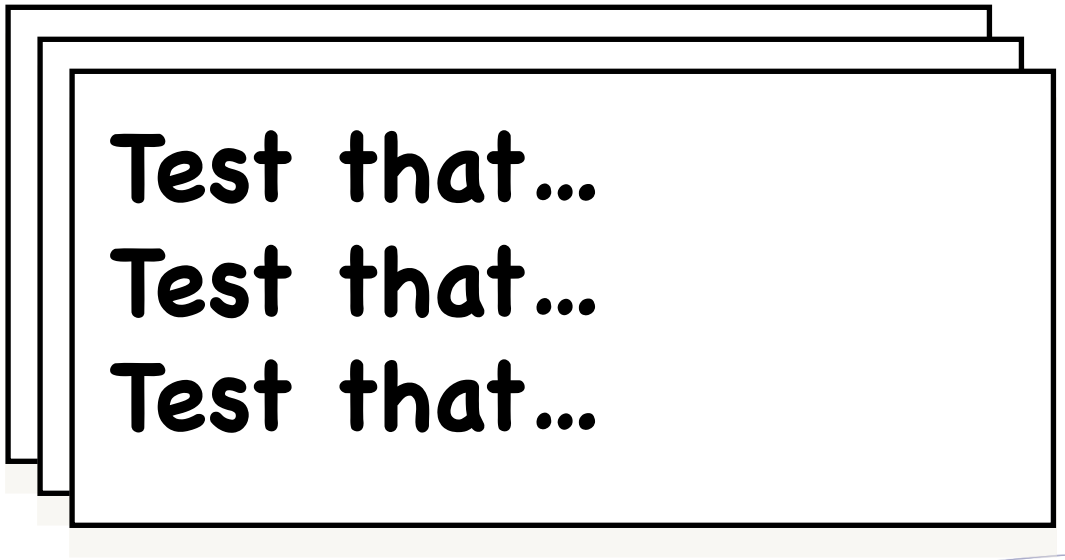
XP - Project Level



Exercise – Acceptance Tests

In teams, for the given project:

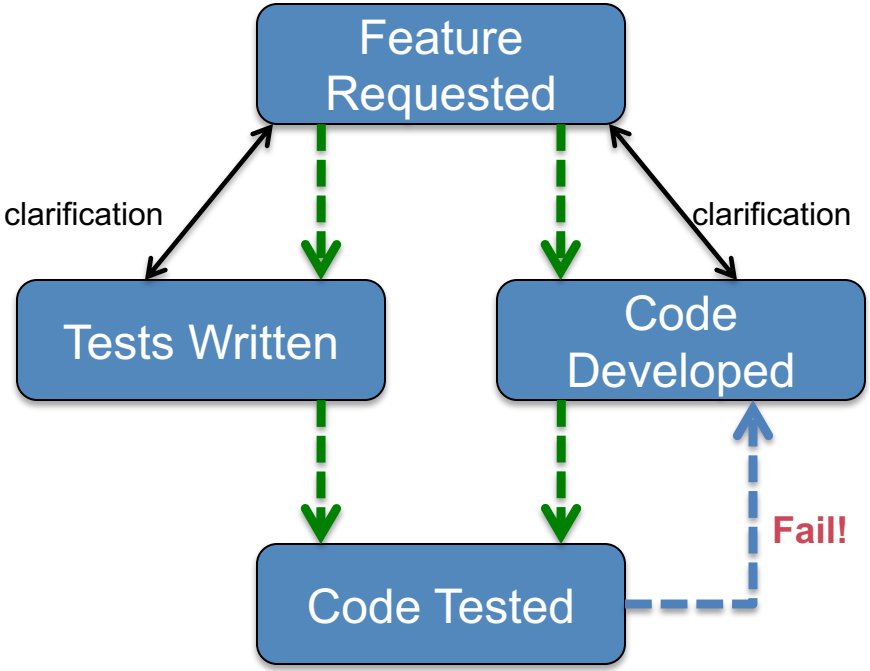
Define acceptance tests for the top **3** stories

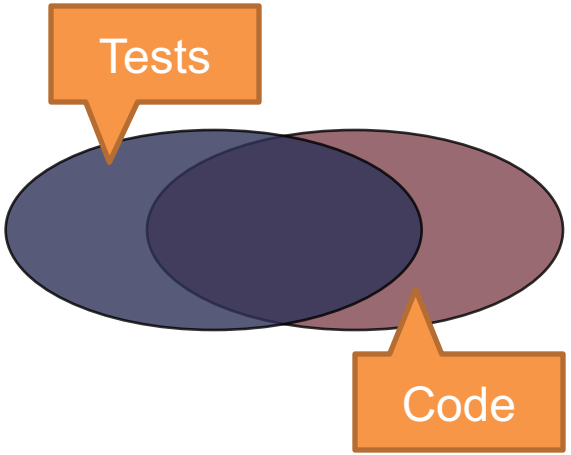


Question

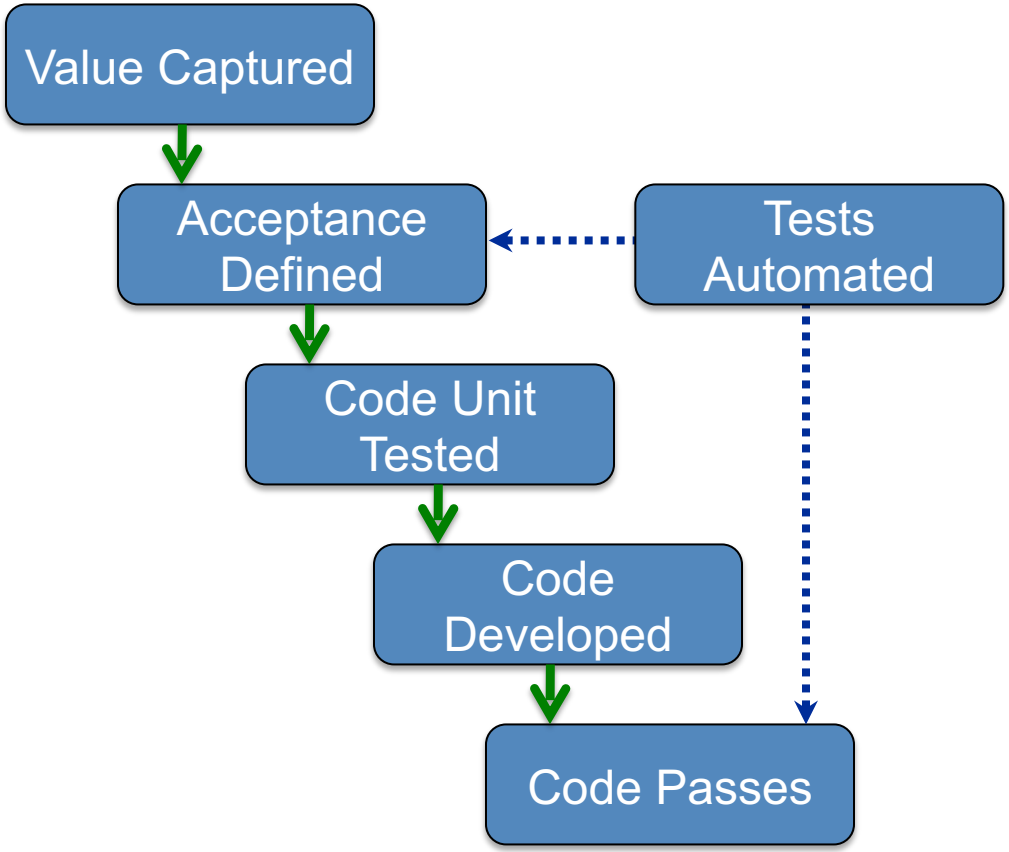
What is the difference between a test and a requirement?

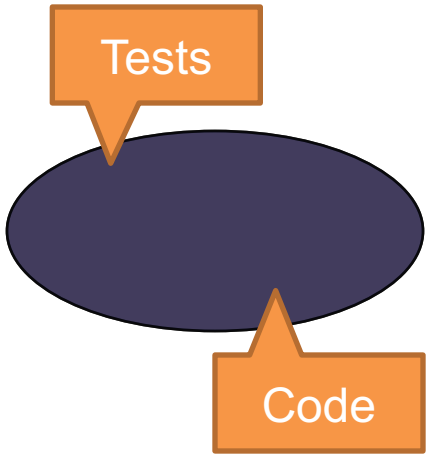
The Role of QA





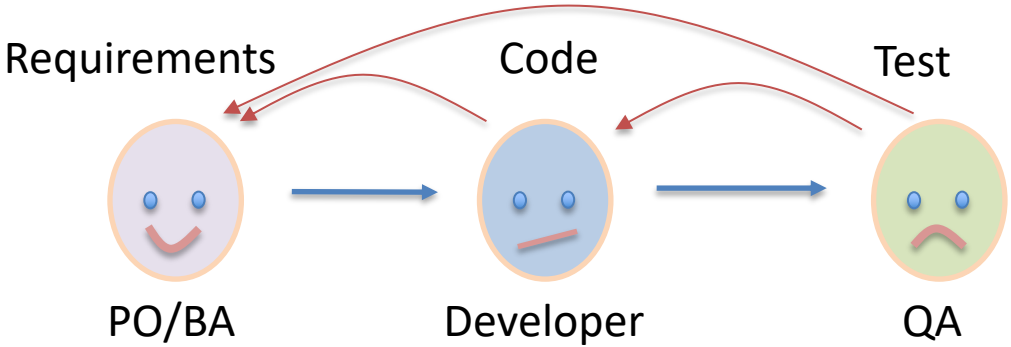
Agile QA


Value Captured
Acceptance Defined
Tests Automated
Code Unit Tested
Code Developed
Code Passes

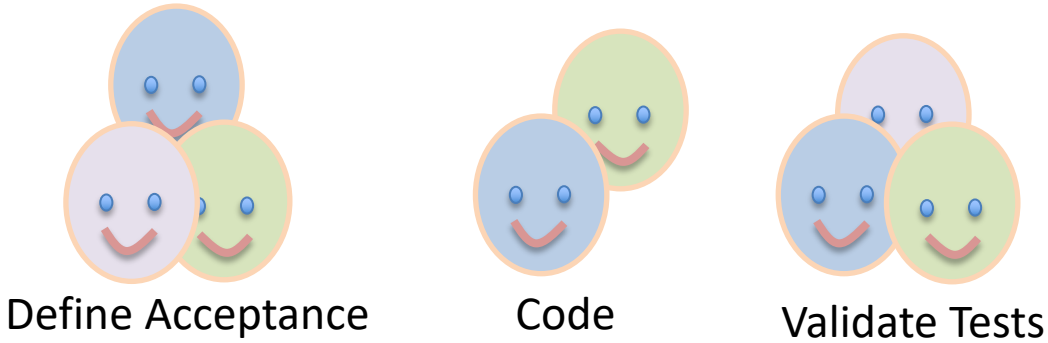

Tests
Code

Why Test First?

Test Last

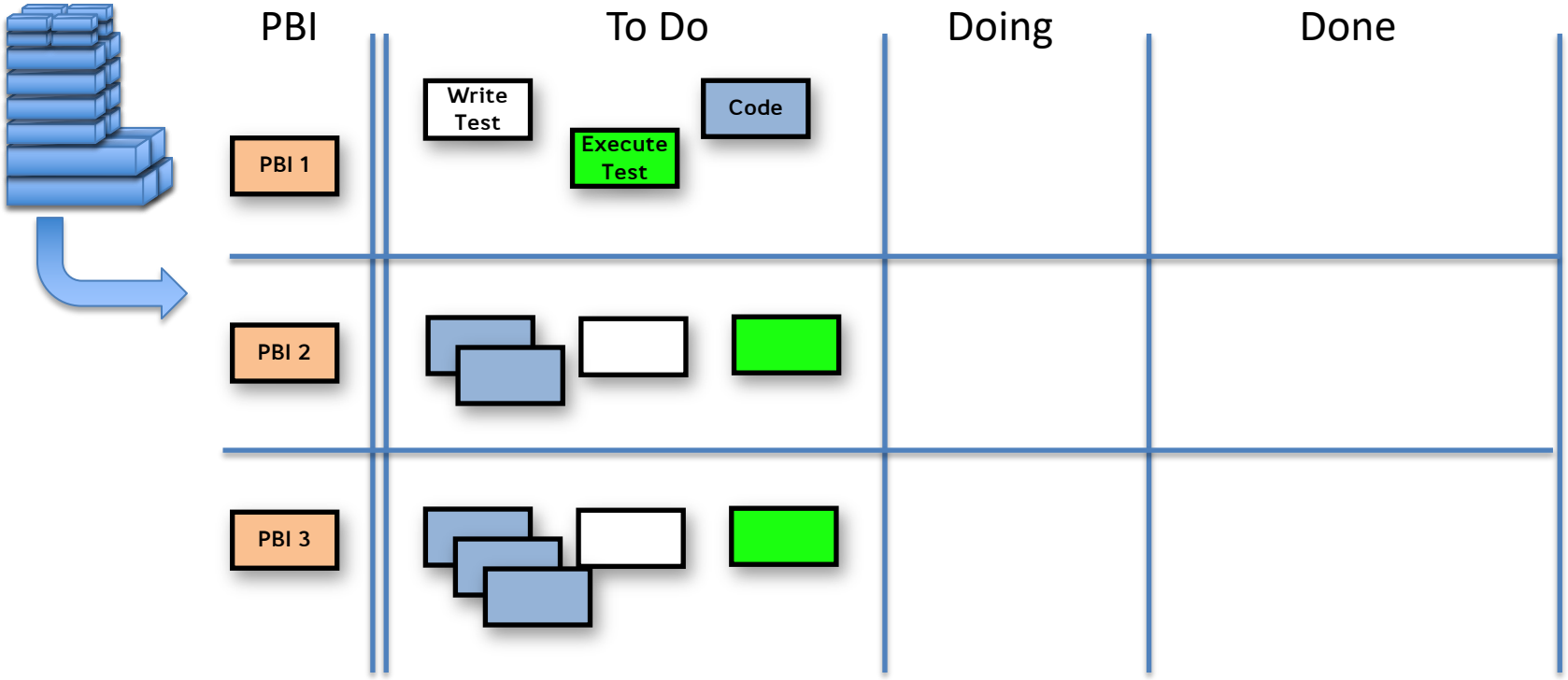


Test First

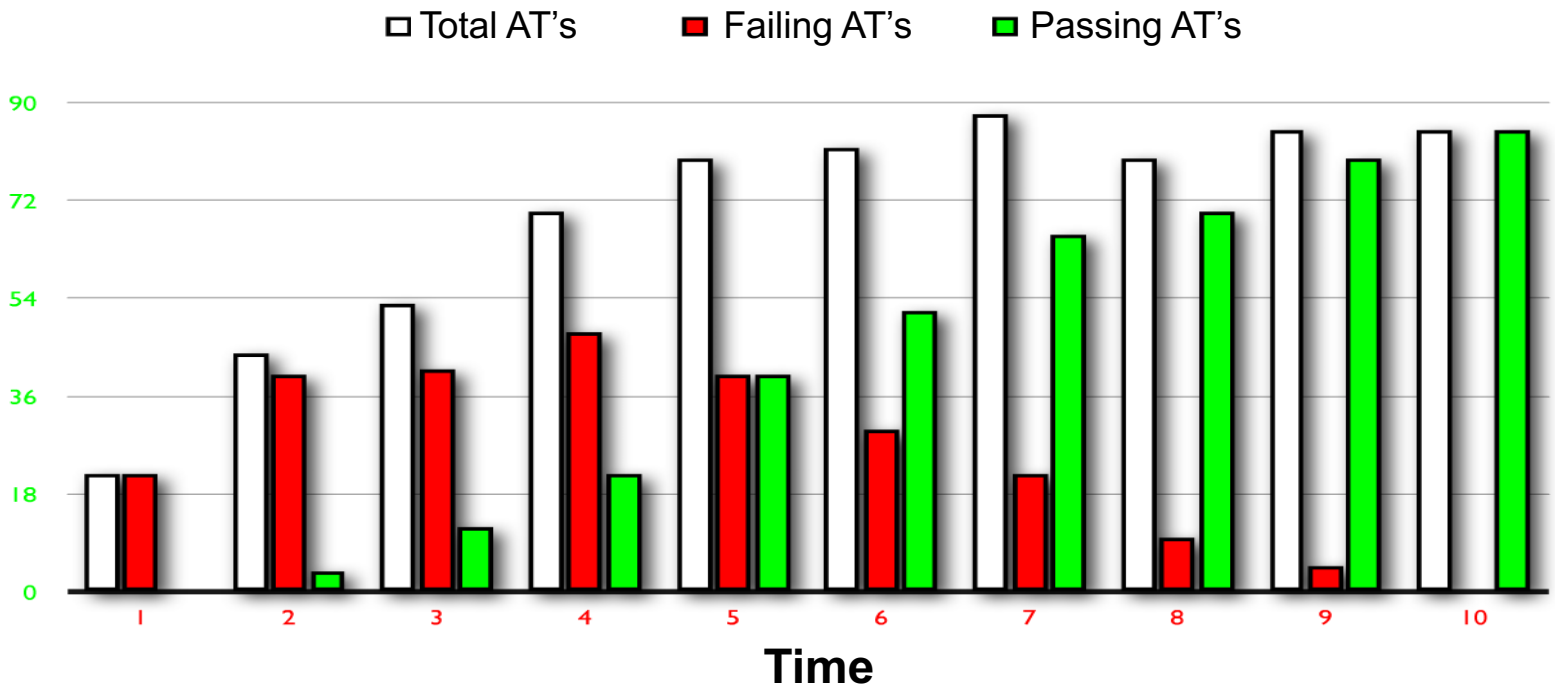


Fewer handoffs. More collaboration.

A day in the life…



The Goal

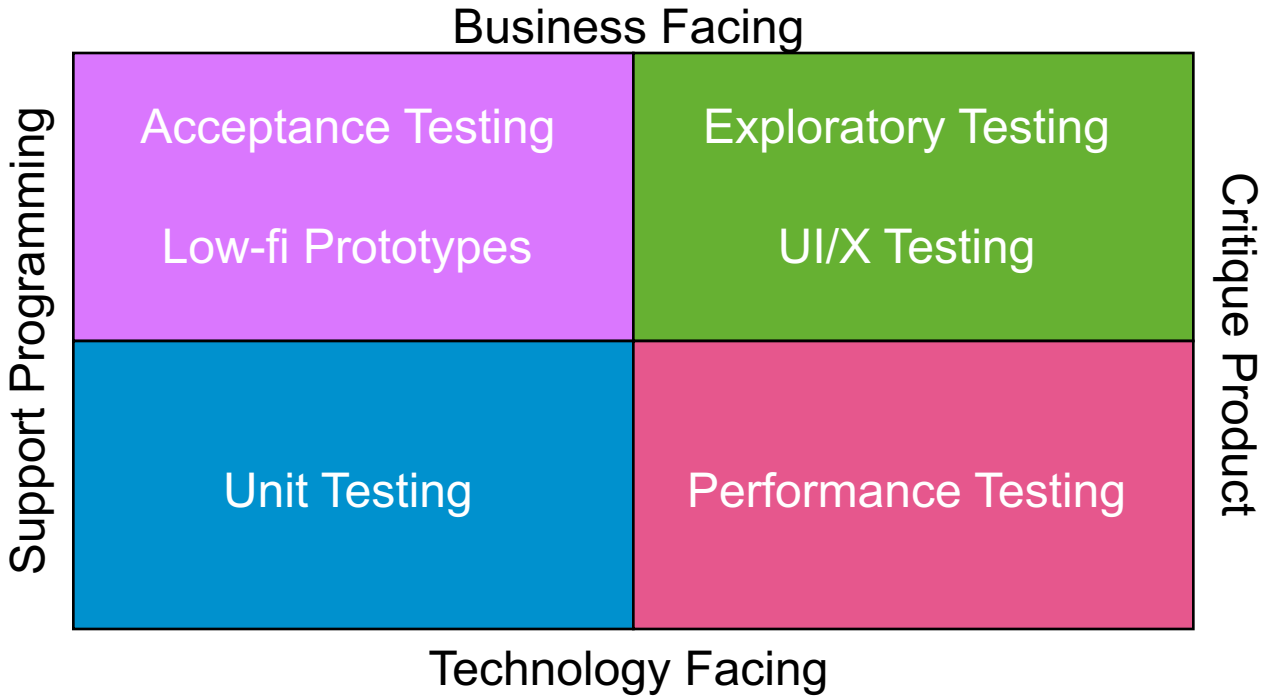

Total AT's
Failing AT's
Passing AT's
90
72
54
36
18
0
1
2
3
4
5
6
7
8
9
10
Time

Setting Expectations...

Acceptance Criteria define what the customer will see in order to approve the work as being complete.

They can be written as test cases or something less detailed.

Test Categorization - Brian Marick



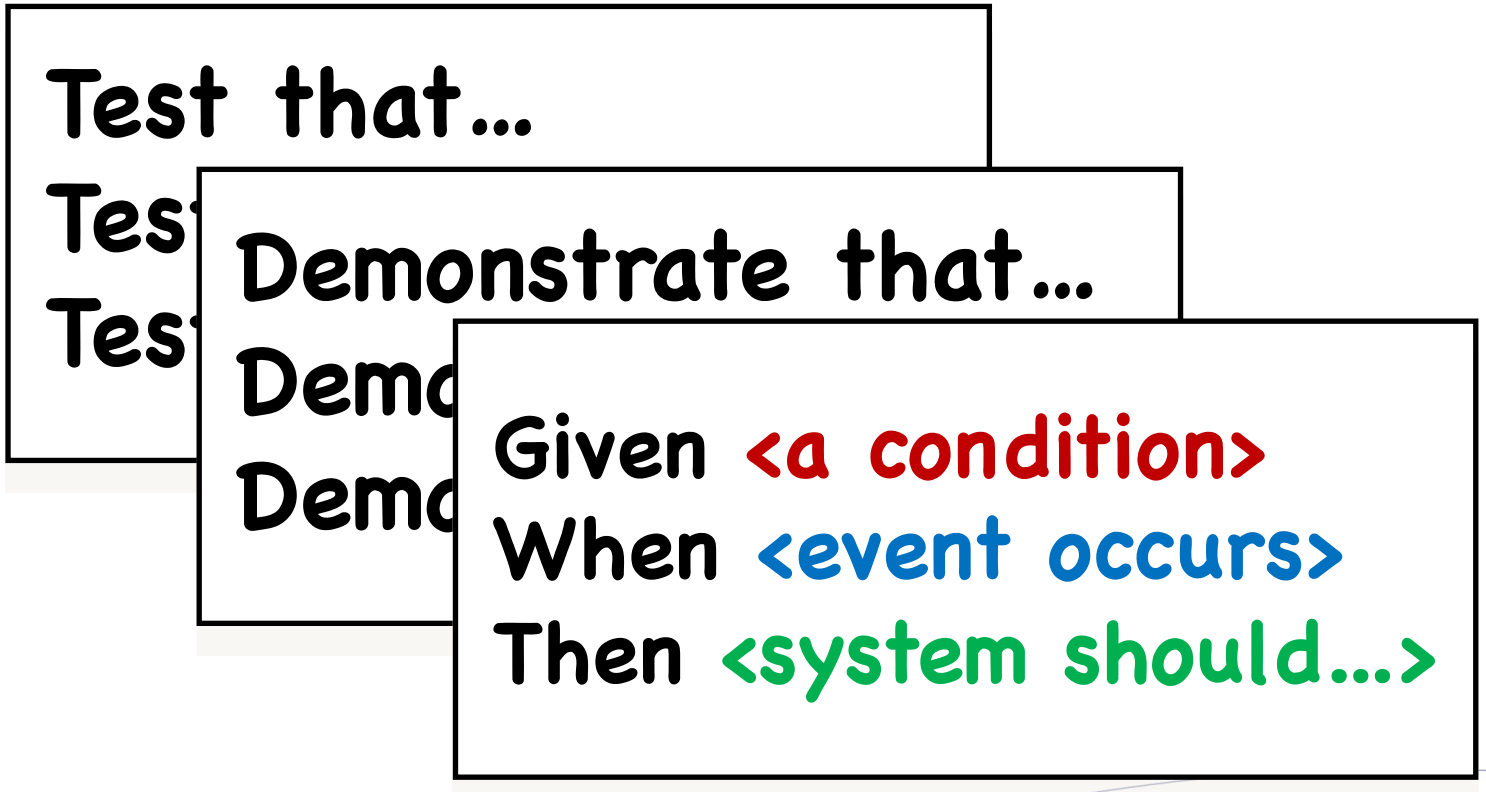
Test Cases

Each test case describes in a step-by-step manner a specific interaction with the system and the expected response.

Test 1:

1. The user visits the home page.
   The home page displays a banner, …, search text box, … .
2. The user searches for "O'Dell".
   The search results page with 1 book, Object-Oriented Methods, should display.
3. The user selects to buy Object-Oriented Methods.
   …

Acceptance Criteria Templates



Acceptance Tests are…

Success

Advance

Fail

Error

Role of Agile QA

- Help define stories
- Ensure stories are testable
- Add stories related to non-functional requirements (usability, performance, etc)
- Organize non-functional testing
- Help define acceptance criteria for stories before they are completed
- Verify the completion of stories as they are completed
- Encourage and/or help developers write good unit tests
- Increase code coverage by adding more automated tests
- Perform exploratory testing on early builds

improving
It's what we do. ™

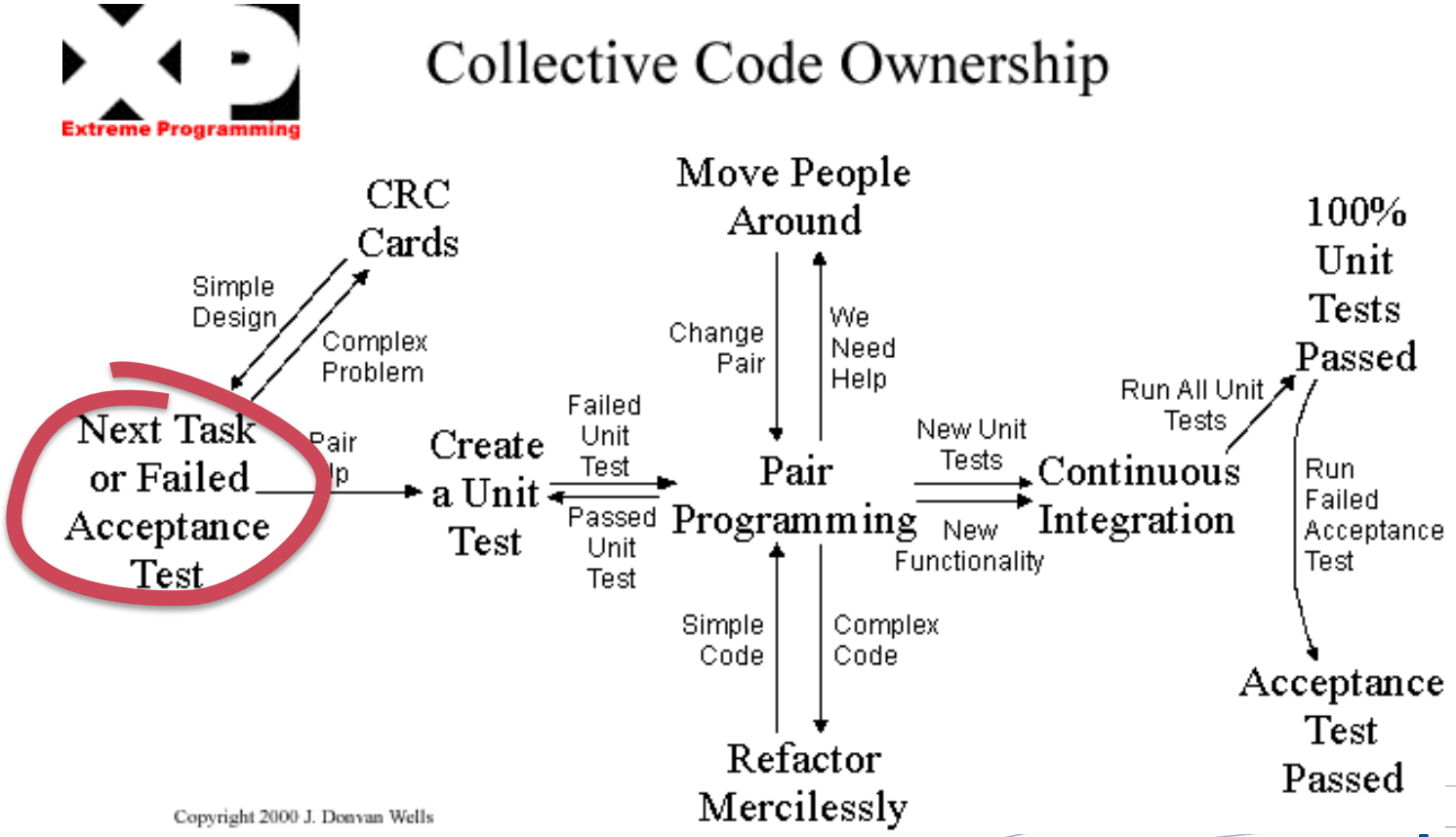
Module 6:

EXAMPLES AS REQUIREMENTS

You will learn to

- Recognize how example specifications set expectations
- Translate tests into examples
- Identify tools for test automation

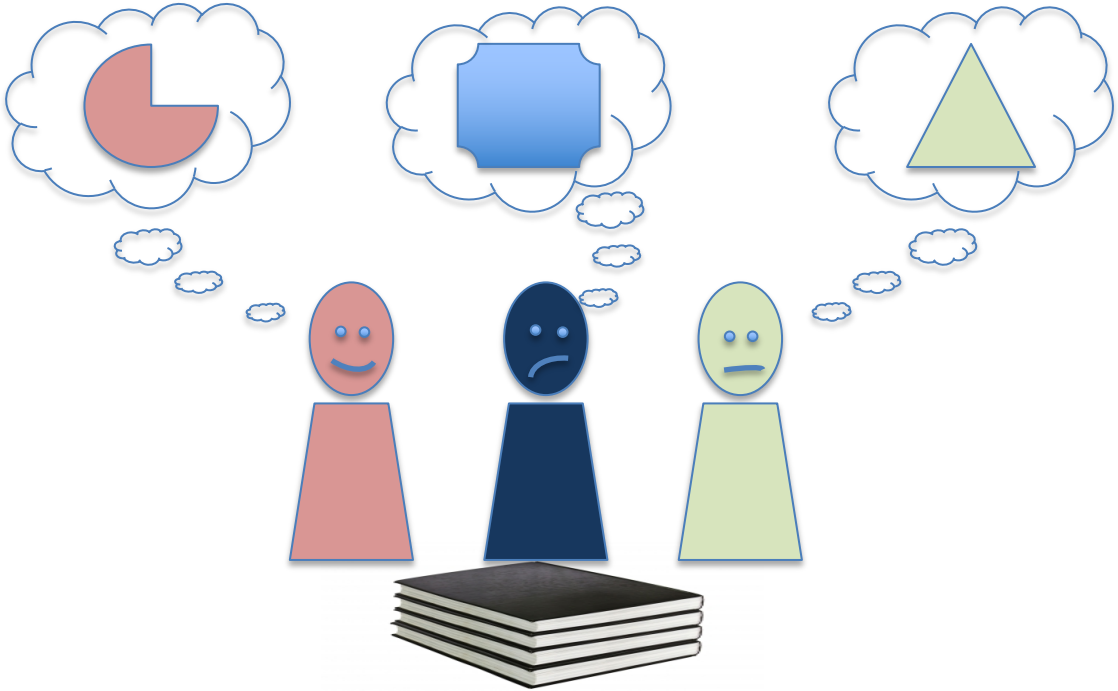
XP – Code Level



Work item handoff
doesn’t work the way you think.

Especially if it’s written

With a document, we may be believe we understand



I’m glad we all agree

*inspired from Jeff Patton
http://www.slideshare.net/jeffpatton/user-story-mapping-discovery-the-whole-story

People misinterpret Text

*Cakewrecks.com

Even what might seem obvious

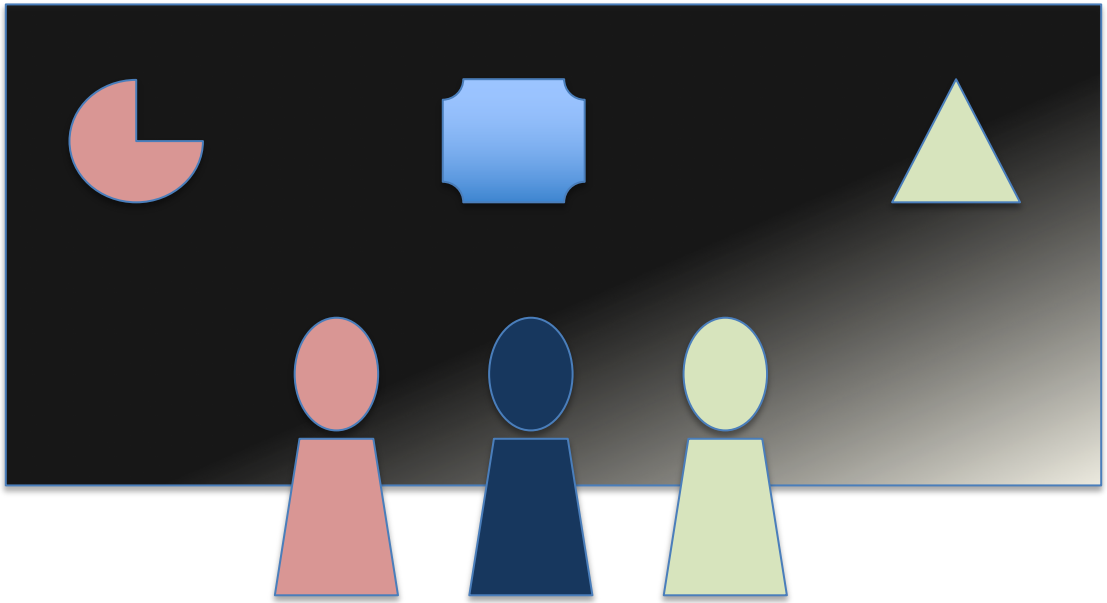
*cakewrecks.com

Each person forms their own idea of what text means

*cakewrecks.com

Text for specification doesn't work well

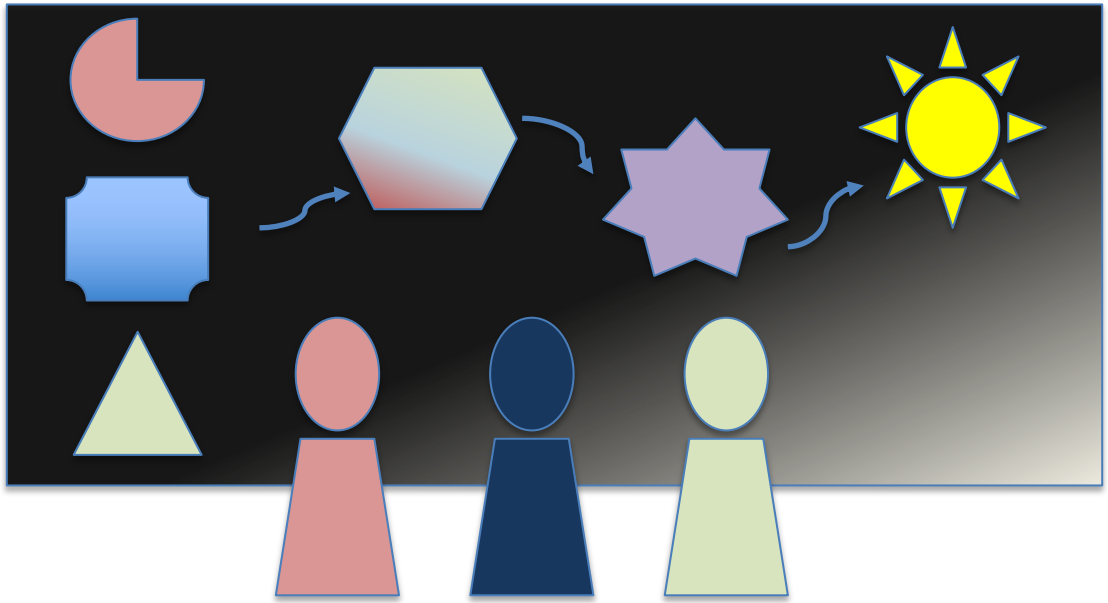
Externalize thinking with conversation and pictures to detect differences



Huh?…

*inspired from Jeff Patton
http://www.slideshare.net/jeffpatton/user-story-mapping-discovery-the-whole-story

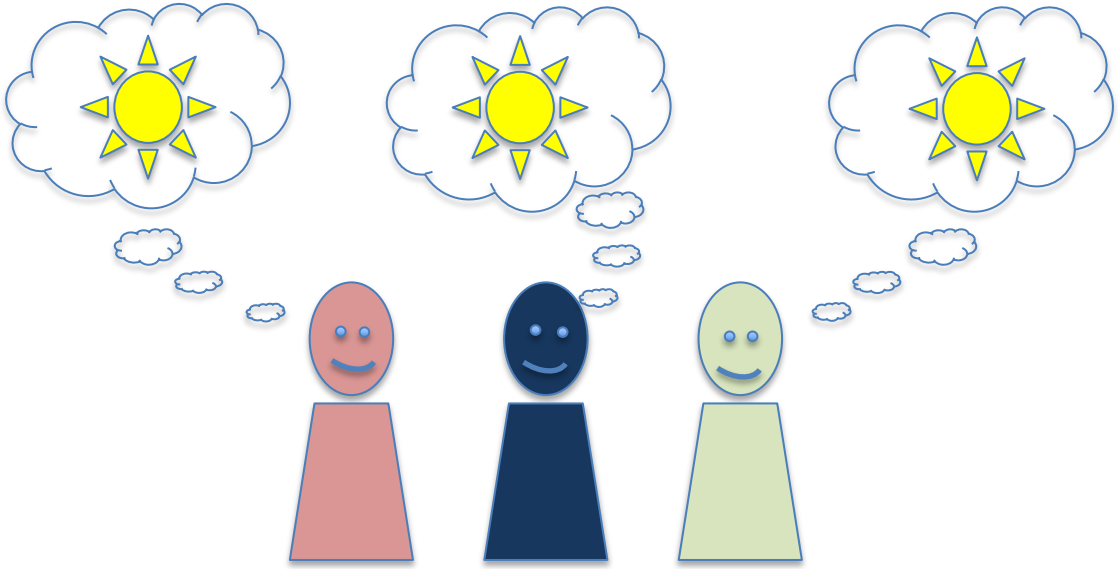
Combine, and Refine to make it better



That's it!

*inspired from Jeff Patton
http://www.slideshare.net/jeffpatton/user-story-mapping-discovery-the-whole-story

Afterwards we have a shared understanding



I’m glad we all agree

*inspired from Jeff Patton
http://www.slideshare.net/jeffpatton/user-story-mapping-discovery-the-whole-story

Shared understanding and alignment are the objectives of collaborative work

*Jeff Patton http://www.slideshare.net/jeffpatton/user-story-mapping-discovery-the-whole-story slide 25

Examples

- Documents the system contract
- An explicit list of
  - Preconditions or given
  - Input/actions or when
  - Outputs/state or then
- Developed as a team to confirm a shared understanding.

Text Specification by Example

Typical Acceptance Test

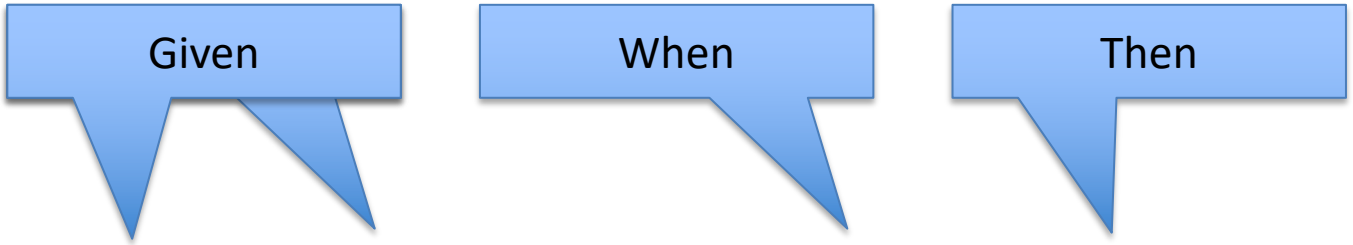
Given a driver has a toll tag
When they drive through a toll
Then the charge is member rate

Given a driver is not a member
When they drive through a toll
Then the charge is non-member rate

Specification by Example

Given member #4356 exists
When they drive through toll SPRNG-CRK
Then the charge is 4 dollars

Given a driver is not a member
When they drive through toll SPRNG-CRK
Then the charge is 6 dollars



Membership Level	Rate Level	Toll Passed	Toll Charge?
None	Normal	SPRNG-CRK	$6
None	Normal	LGCY	$6
Silver	Normal	SPRNG-CRK	$4
Gold	Normal	SPRNG-CRK	$2
None	Peak	SPRNG-CRK	$10
Silver	Peak	SPRNG-CRK	$6

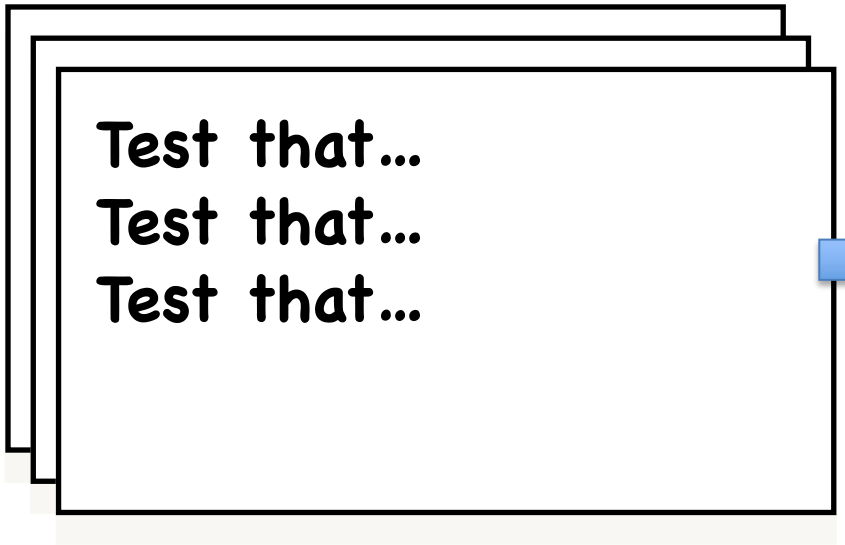
Exercise

- Create specification by example for TTD Airlines
- Register Member
- Collect
  - User ID
  - Email

Exercise – Acceptance Tests

In teams, for the given project:

Translate acceptance tests into examples

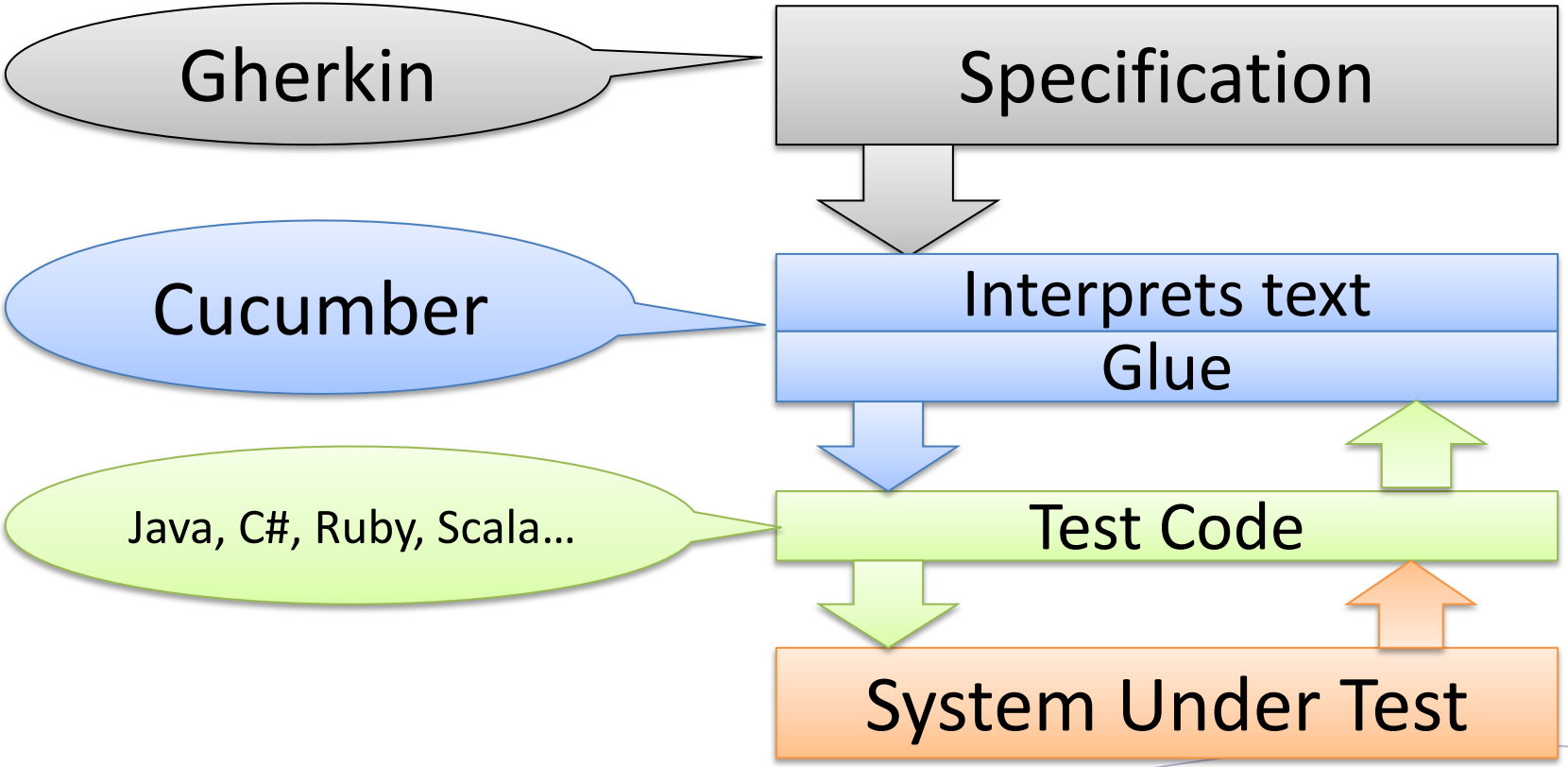


Level	Rate Level	Toll Passed	Toll Charge?
None	Normal	SPRNG-CRK	$6
None	Normal	LGCY	$6
Silver	Normal	SPRNG-CRK	$4
Gold	Normal	SPRNG-CRK	$2
None	Peak	SPRNG-CRK	$10
Silver	Peak	SPRNG-CRK	$6

Acceptance Testing Automation Tools

- Fit and FitNesse
- Cucumber
- Concordian
- JUnit?

Gherkin VS Cucumber



Gherkin looks like this:

```
1: Feature: Some terse yet descriptive text of what is desired
2:   Textual description of the business value of this feature
3:   Business rules that govern the scope of the feature
4:   Any additional information that will make the feature easier to understand
5:
6:   Scenario: Some determinable business situation
7:     Given some precondition
8:       And some other precondition
9:      When some action by the actor
10:      And some other action
11:      But yet another action
12:    Then some testable outcome is achieved
13:      And something else we can check happens too
14:
15:  Scenario: A different situation
```

keywords

Informative text for humans and tools

Steps that glue to test code

Gherkin Specification

tollCalculator.feature

```
Feature: Toll Tag Cost
As a Driver
I want to know the cost of taking a toll road
In order to plan travel

Scenario: Calculate the toll for 10 miles
When I drive my vehicle on the toll road for 10 miles
Then I will have to pay $20

Scenario: Calculate the toll when premium rates are in effect
Given premium rates are in effect
And I have a "silver" level toll tag
When I drive my vehicle on the toll road for 10 miles
Then I will have to pay $40
```

Strings in quotes

Code Snippet Generation

Then Run the feature – it should provide the code stub that you can place in a POJO.


```
@When("^I drive my vehicle on the toll road for (\\d+) miles$")
public void i_drive_my_vehicle_on_the_toll_road_for_miles(int arg1) throws
Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@Then("^I will have to pay \\$(\\d+)$")
public void i_will_have_to_pay_$(int arg1) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}
```

Using Variables

Scenario Outline: Withdraw fixed amount
Given I have <Balance> in my account
When I choose to withdraw the fixed amount of <Withdrawal>
Then I should <Outcome>
And the balance of my account should be <Remaining>

Examples: Successful withdrawal

Balance	Withdraw	Outcome	Remaining
$500	$50	"receive $50 cash"	$450
$500	$100	"receive $100 cash"	$400

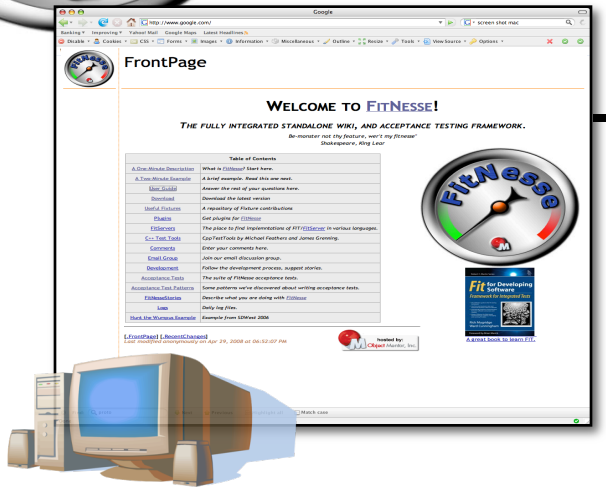
Examples: Attempt to withdraw too much

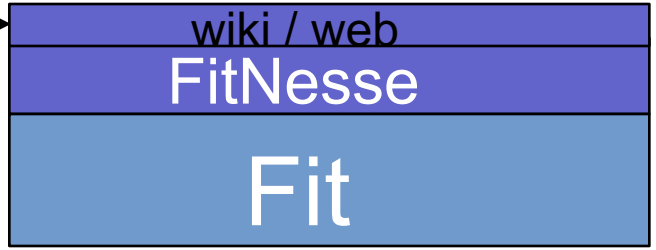
Balance	Withdraw	Outcome	Remaining
$100	$200	"see an error message"	$100
$0	$50	"see an error message"	$0

Example from "The Cucumber for Java Book"

Fit / FitNesse

The fully integrated standalone wiki, and acceptance testing framework.

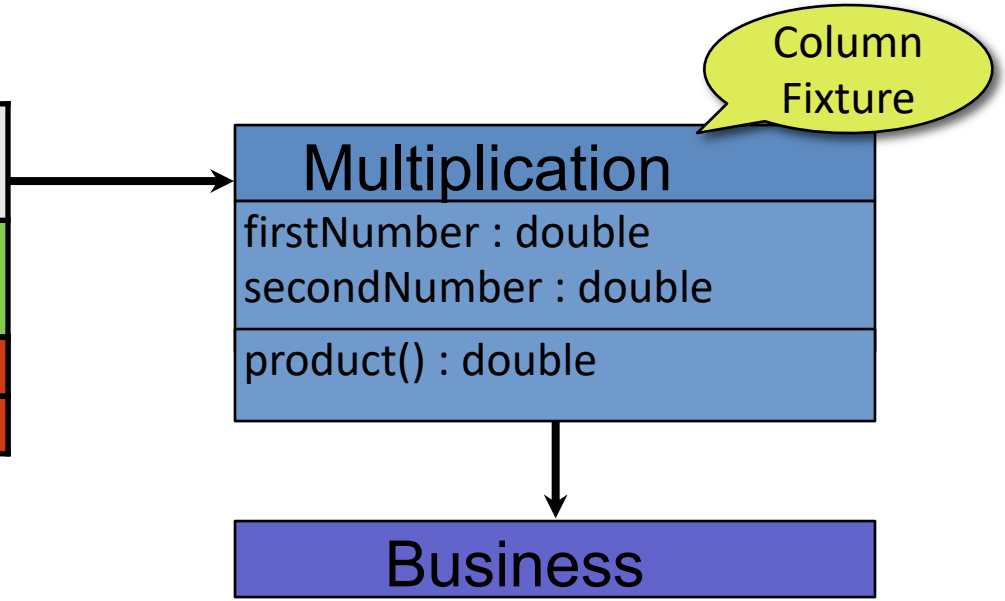




Fit / FitNesse

The fully integrated standalone wiki, and acceptance testing framework.

first number	second number	product?
10	4	40
12.3	5	61.5 *expected* 61 *actual*



Fit / FitNesse

The fully integrated standalone wiki, and acceptance testing framework.

Multiplication

first number	second number	product?
10	4	40
12.3	5	61.5 *expected* 61 *actual*

Column Fixture

```
public class Multiplication extends ColumnFixture {

  public double firstNumber;
  public double secondNumber;

  public double product() {
    Calculator calculator = new Calculator();

    double result =
      calculator.multiply(firstNumber, secondNumber);

    return result;
  }
}
```

Using a xUnit Tool for BDD

```
public class WhenCancellingReservation{

    [TestMethod]
    public void ShouldNotChargeFeeForPremium() {
        …
    }

    [TestMethod]
    public void ShouldCharge10PercentForNonPremium() {
        …
    }

    [TestMethod]
    public void ShouldSendConfirmationEmail () {
        …
    }

    [TestMethod]
    public void ShouldNotifyHotels() {
        …
    }

}
```

As a vacationer,
I want to cancel a reservation so that I can adjust my plans.

- ❑ Verify that a premium member can cancel the same day without a fee.
- ❑ Verify that a non-premium member is charged 10% for a same-day cancellation.
- ❑ Verify that an email confirmation is sent.
- ❑ Verify that the hotel is notified of any cancellation.

improving
It's what we do. ™

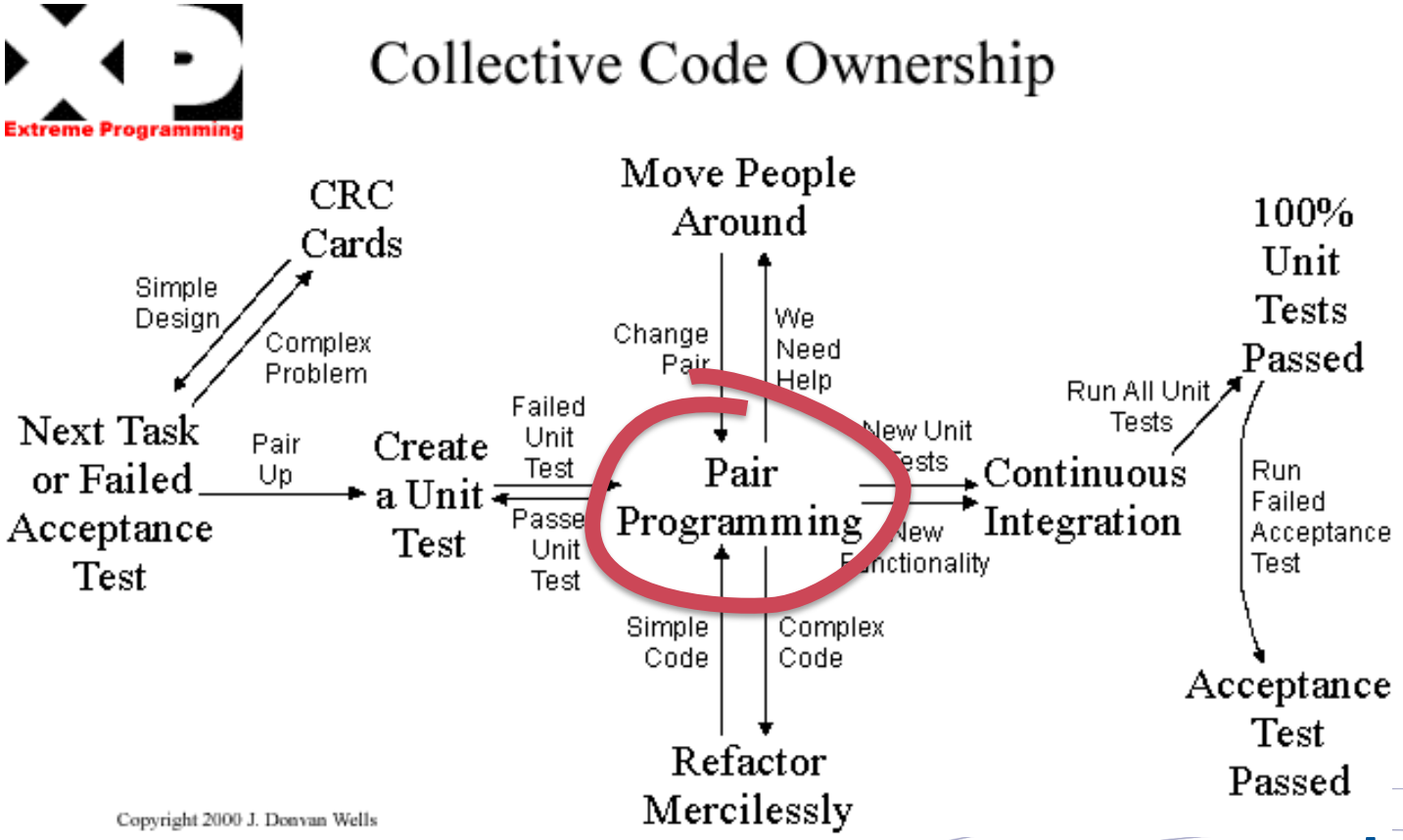
Module 7:

PAIR PROGRAMMING

You will learn to

- Recognize the benefits of Pair Programming
- Implement Pair Programming techniques
- Recognize the advantages of Mob Programming

XP – Code Level



Pair Programming Behaviors

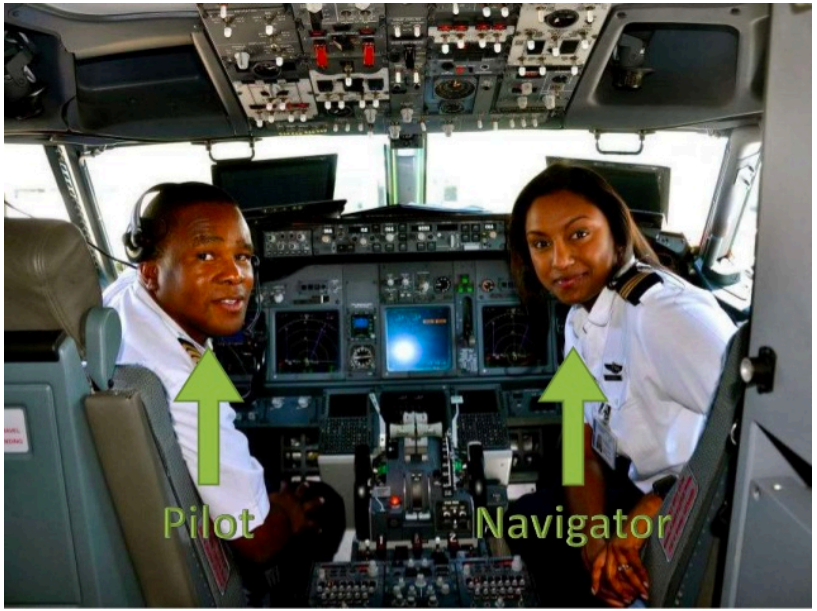
1. **Pair Pressure**
   Positive encouragement
2. **Pair Negotiation**
   Combining two sets of skills
3. **Pair Courage**
   Trying together what you might not alone
4. **Pair Review**
   Continuous review
5. **Pair Debugging**
   More enjoyable and shorter
6. **Pair Learning**
   Acquiring/improving skills
7. **Pair Trust**
   Building confidence in team mates

Classic Pairing

One person directing like a navigator

One person typing like the pilot

Pilot can advise the navigator, but navigator has final call on direction.



One person typing silently is not pair programming!

Pairing as Mentoring

With mixed skills, it should become mentoring

- Requires correct attitude from more experienced.
- In time crunch,
  - May not be the right time.
  - May be the perfect time.

Getting into the flow

- This is hard!
  - My head hurts until I get back into the flow
- Makes the day flow once you are use to it.

Mob Programming

Use pairing as start of a mob.

Single → Pair → Mob

Limits work in progress.

Ping Pong Pairing

- Use 2 computers
- Make changes on one computer at a time.

1. Navigator directs Pilot to write a failing test
2. Commit and Push test into the repository
3. Swap computers and roles
4. Pull to get update
5. Navigator directs Pilot to make failing test **barely** pass
6. Refactor together
7. Repeat

improving
It's what we do. ™

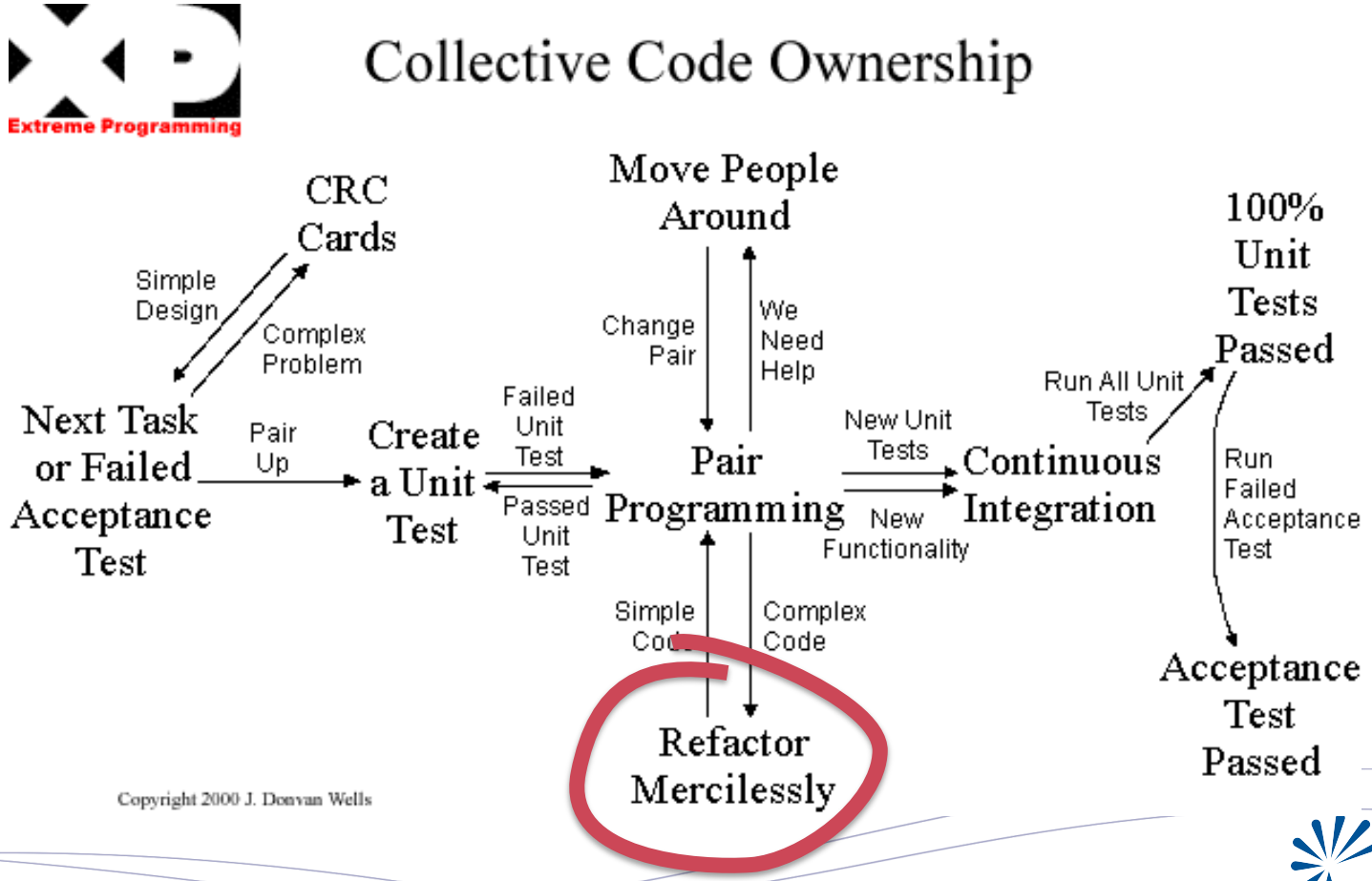
Module 8:

TEST DOUBLES

You will learn to

- identify types of test doubles: dummies, stubs, spies, fakes, and mocks
- recognize how test doubles make unit tests independent and fast
- distinguish between integration and unit tests

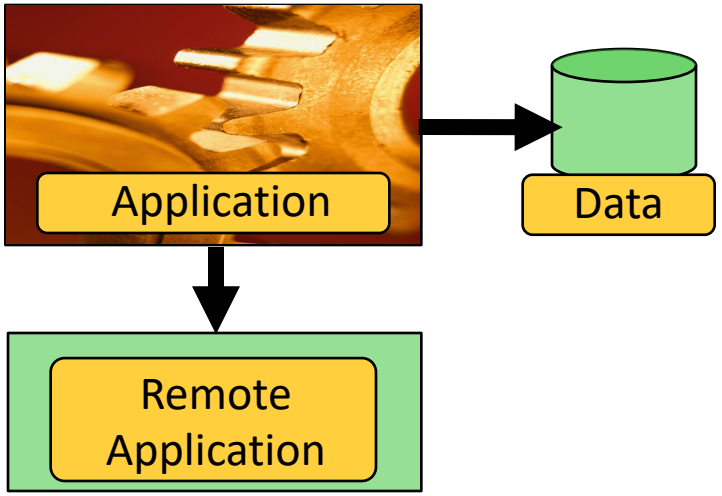
XP – Code Level



The Problem

- External dependencies affect
  - Performance
  - Portability
  - Stability

... of your tests.

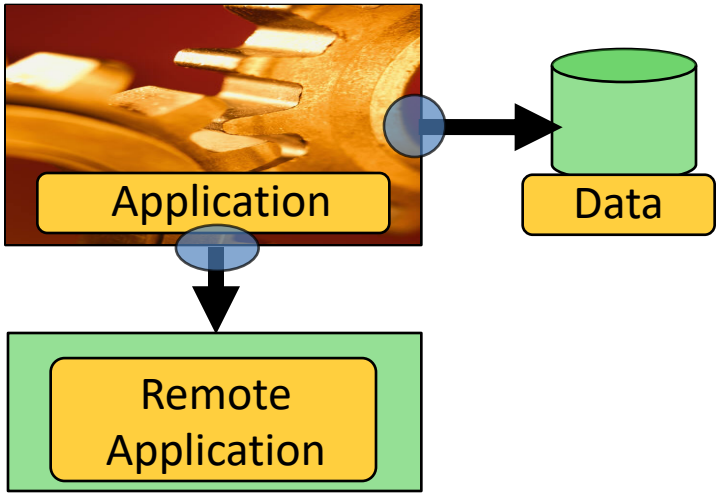


This is an opportunity and a driving force to design more modular, more testable and more re-usable code.

Integration Tests

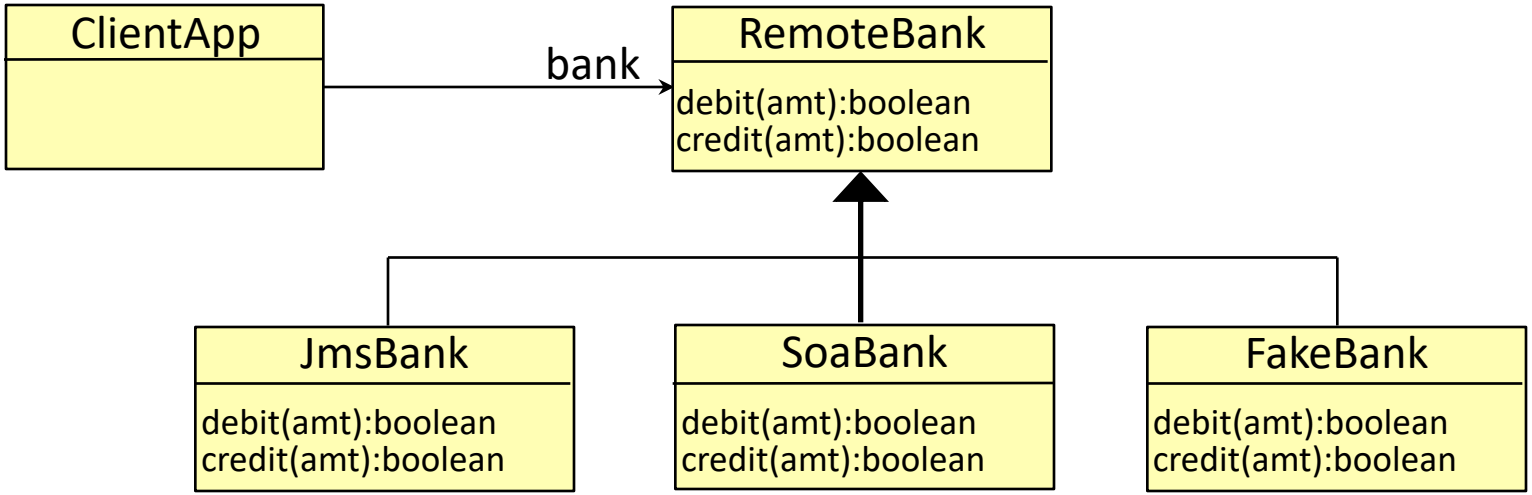
- Testing external connections to other systems and databases are acceptable, but not as part of the general test suite.

- These are called **Integration Tests**



The Solution

Dependency Inversion



Set up Dependency Injection for Doubles

- Be able to set the dependence through the constructor or a setter – maybe a factory

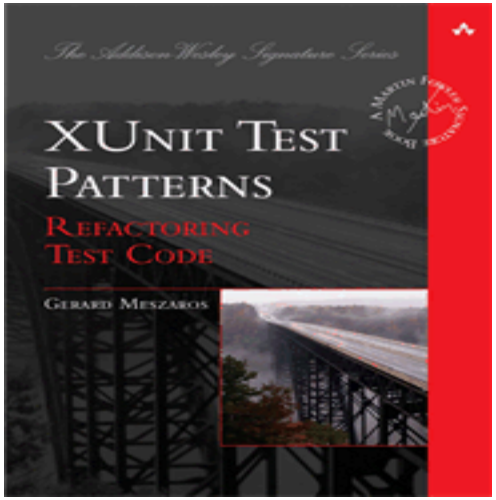
```
public class ClientApp{
  public ClientApp( Persistor persistor, RemoteBank externalProxi){…}
}
```

- Inject Doubles when doing tests

```
@Test
void whenThereIsSomeTesting(){
     ClientApp subject = new ClientApp(
          new PersistorDouble(), new FakeBank() );
     subject.doSomething();
}
```

XUnit Patterns

- **Test Doubles:**
  - dummies
  - stubs
  - fakes
  - mocks
  - spys



http://xunitpatterns.com

Dummy Object

- Made up values used because they are necessary to invoke a method
- Otherwise are not important for the tests

- Null, "Ignored String", new Object()

```
public class ItemBehavior {

    @Test
    public void shouldCheckIfInventoryIsLow() {
        Item item = new Item("name", 0.0, 10);

        item.setQuantity(20);
        assertFalse(item.isLowIn

        item.setQuantity(2);
        assertTrue(item.isLowInve
    }

    @Test
    public void shouldStoreItem() {
        Item item = new Item("iPhone"
        Store subject = new Store();

        subject.add

        assertEquals(1, subject.getIn

}
```

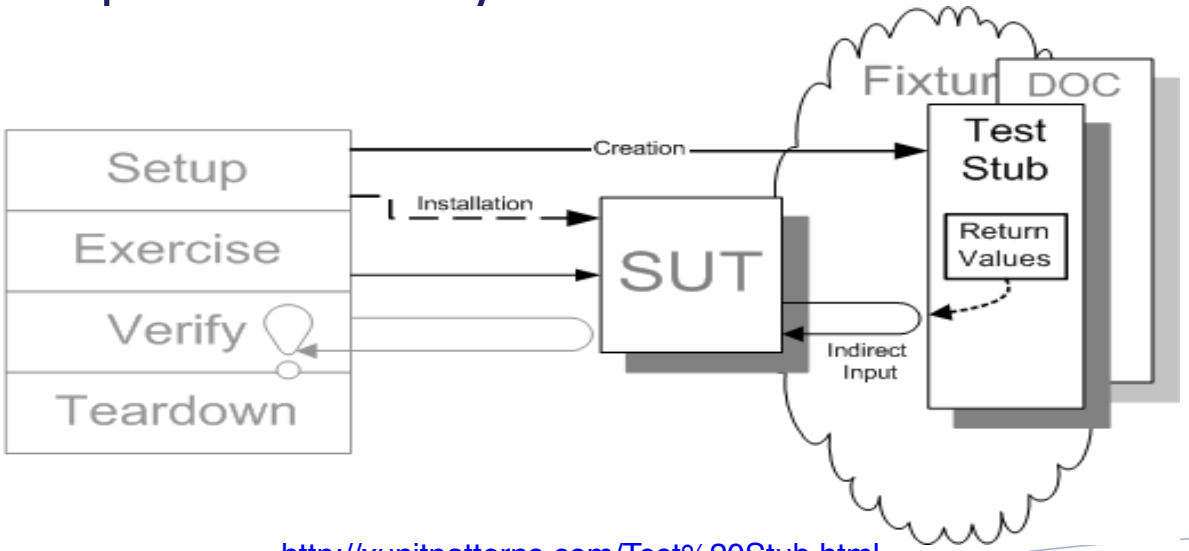
Name and cost don't matter for the test

item is a dummy, we don't care what its values are

Test Stub

```
ClientApp app = new ClientApp();
RemoteBank bank = new FakeBank();
app.setBank(bank);

boolean success = app.debit(1000);
```

- Replace a real object with a test-specific object that feeds the desired inputs into the system under test.

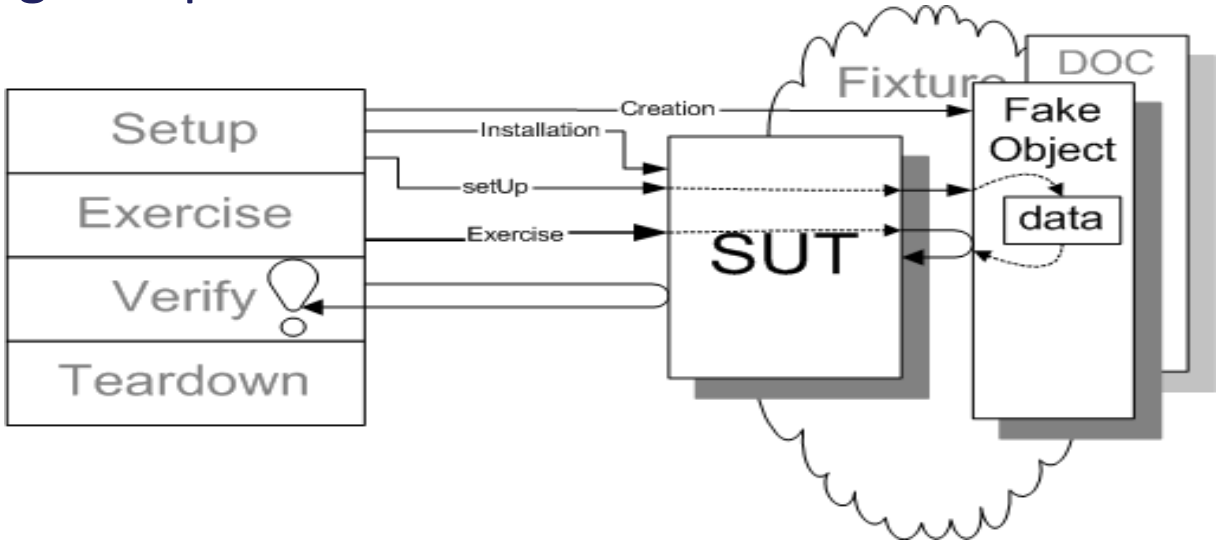


http://xunitpatterns.com/Test%20Stub.html

Fake Object

```
ClientApp app = new ClientApp();
RemoteLogger log = new FakeLogger();
app.setLogging(log);

boolean success = app.debit(1000);
```

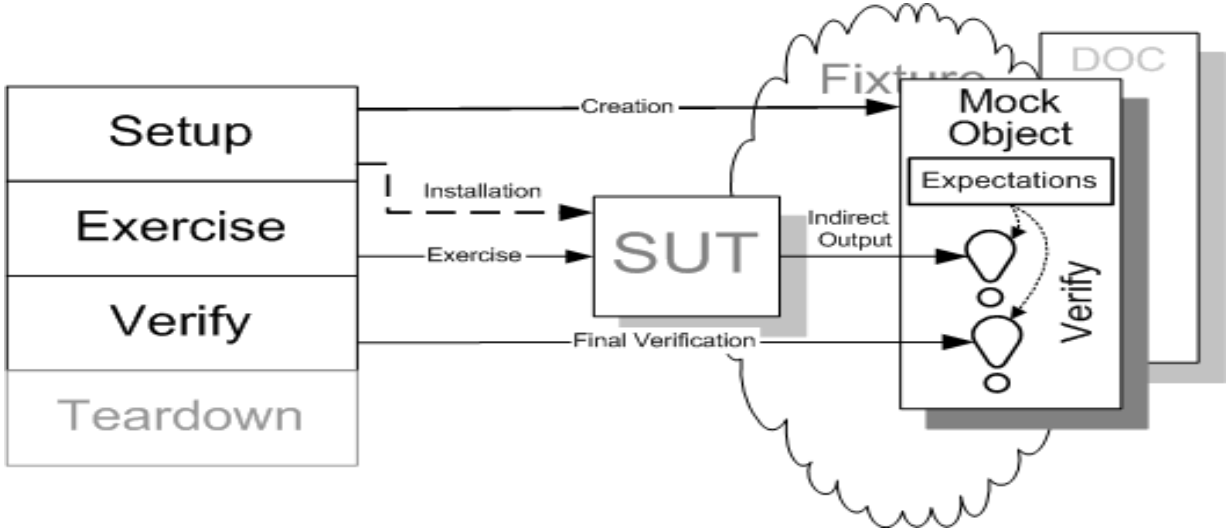
- Replace a component that the SUT depends on with a much lighter-weight implementation.



http://xunitpatterns.com/Fake%20Object.html

Mock Object

- Replace an object the SUT depends on with a test-specific object that verifies it is being used correctly by the SUT.



http://xunitpatterns.com/Mock%20Object.html

Mock Object

```
ClientApp app = new ClientApp();
RemoteLogger log = new MockLogger();
app.setLogging(log);

boolean success = app.debit(1000);
```

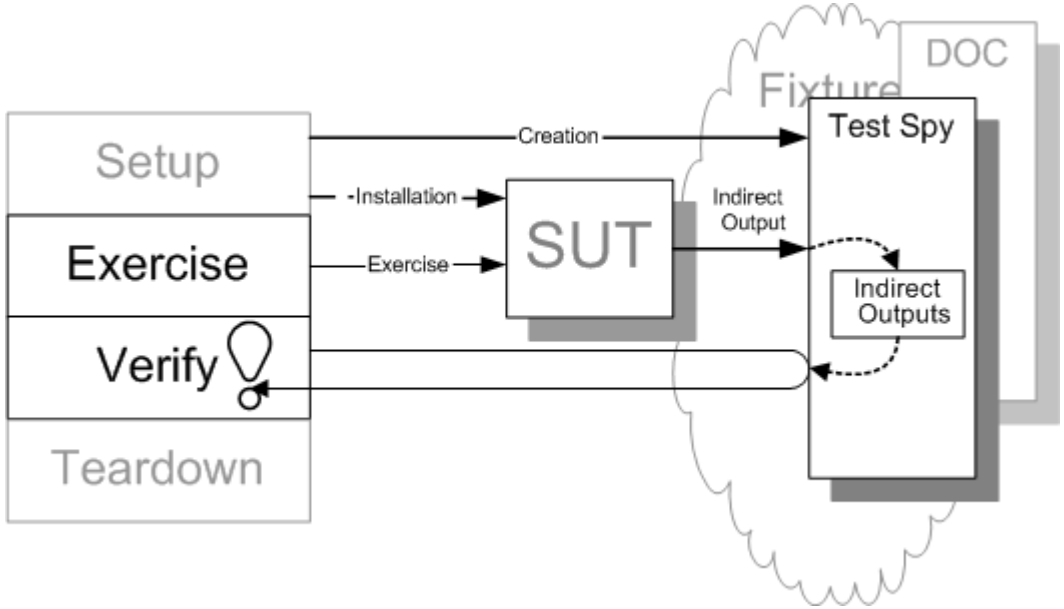
A Mock Logger object:

```
public class MockLogger implements RemoteLogger{

    public void logMessage(Date actualDate, String actualUser,
                           String actualActionCode, Object actualDetail) {
        actualNumberCalls++;

        Assert.assertEquals("date", expectedDate, actualDate);
        Assert.assertEquals("user", expectedUser, actualUser);
        Assert.assertEquals("action code", expectedActionCode, actualActionCode);
        Assert.assertEquals("detail", expectedDetail, actualDetail);
    }

}
```

http://xunitpatterns.com/Mock%20Object.html

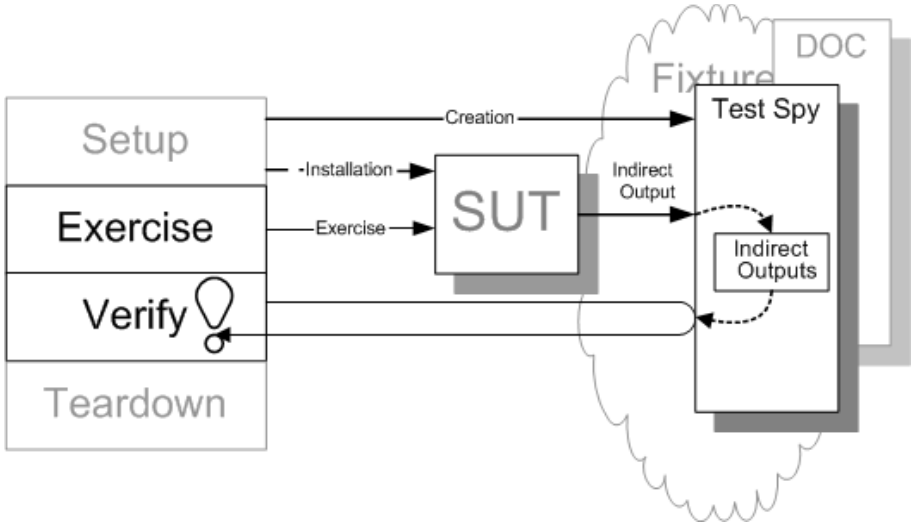
Test Spy

- Substitute Test spy for real object.
- Verify data was received.



Lab - XUnit Patterns

- Apply XUnit patterns.



improving
It's what we do. ™

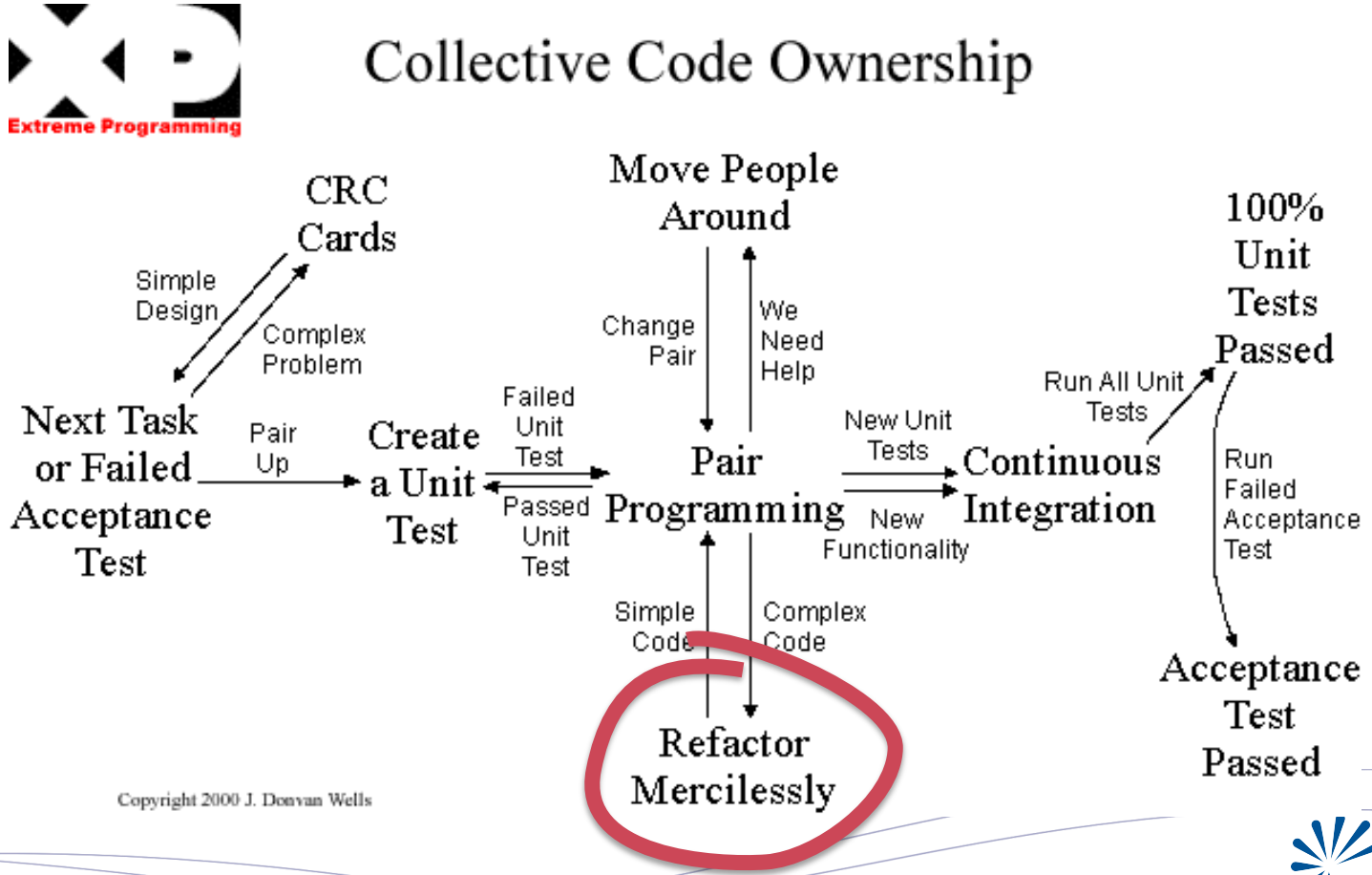
Module 9:

TESTING LEGACY CODE

You will learn to

- Define ‘Legacy’ code
- Describe the steps involved in a Legacy Management Strategy

XP – Code Level

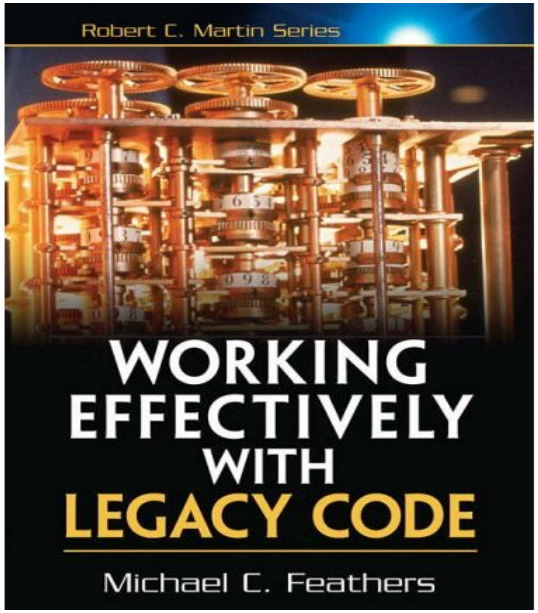


Legacy Code

Working with Legacy Code

Legacy code == code without tests

- It's not about how old the code is



Legacy code change algorithm

- Analyze the impact of change
- Build a safety net
- Change the code
  - Enable the change
  - Add and modify tests for the change
  - Make the change
- Refactor

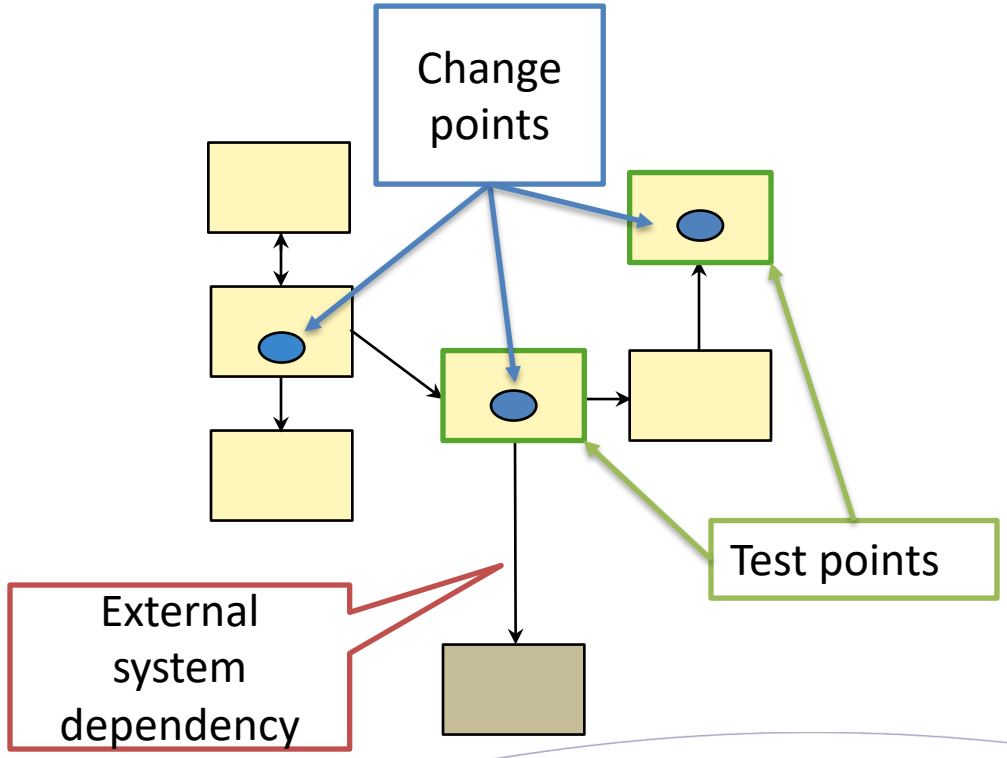
Analyze the impact of the change

For every change:

- Identify Change Points
- Identify Dependencies
- Select Test Points

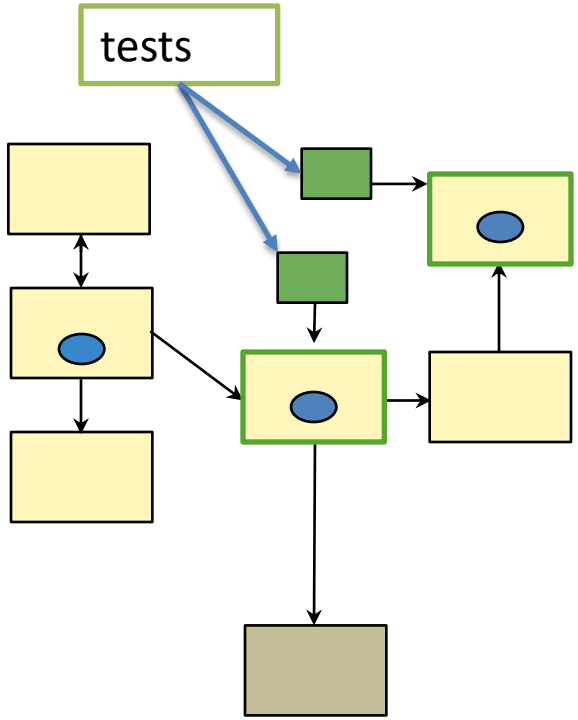
Start with the smallest manageable change.

- Only 2 of 3 change points selected for now



Test First

Write characterization tests



- Only test the selected change points

Make a Safety Net

- Create ‘Characterization’ tests to ensure that no functionality is lost.
  - Document the actual current behavior of a component as opposed to what it ‘should’ do
  - Create tests that pass even for defects
  - Do not attempt to build tests for all existing code
    - Limit tests to change points.

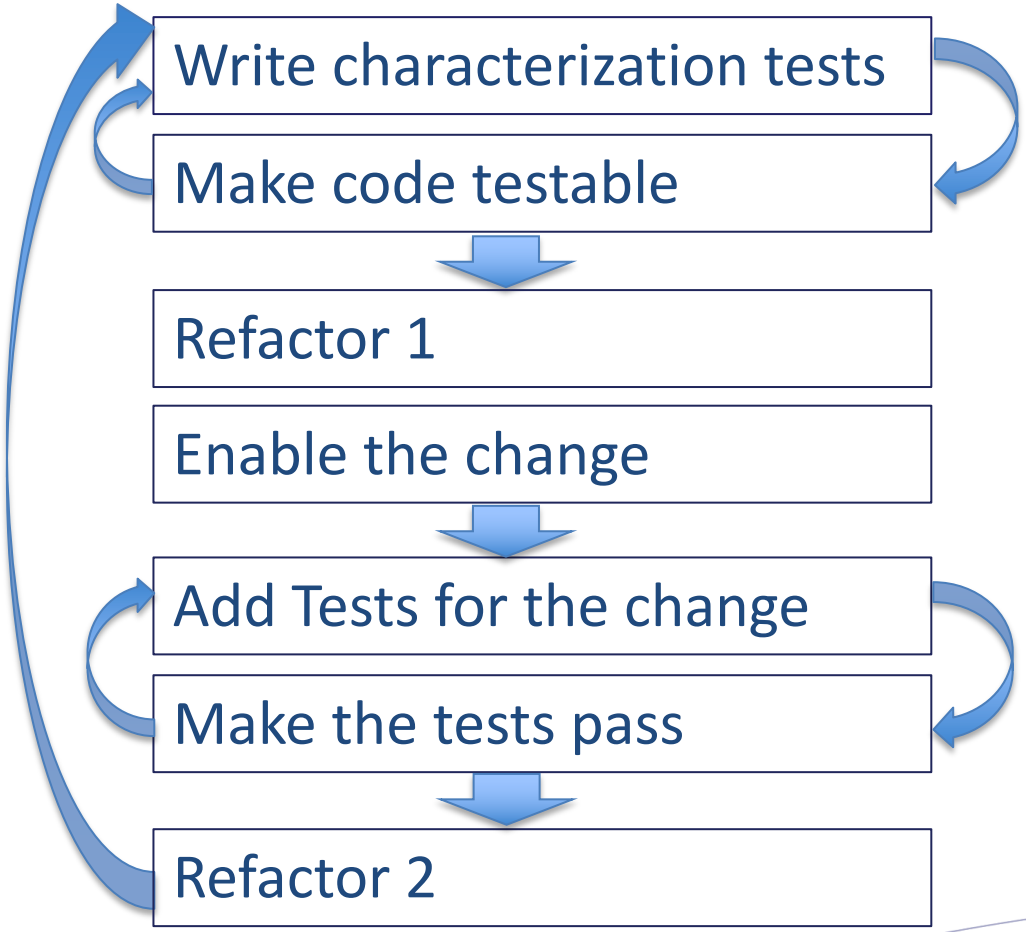
*Tools exist to generate characterization tests automatically

Convert

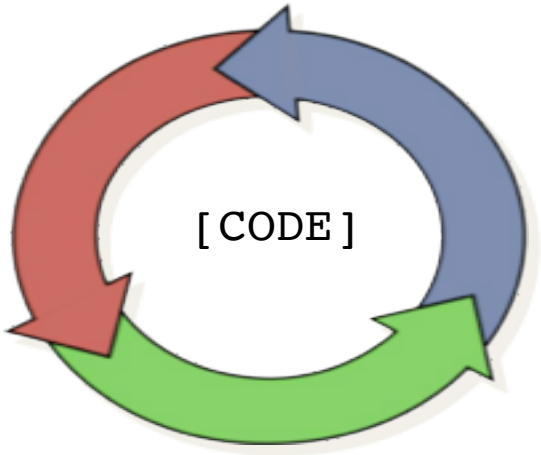
Legacy code

into

Documented code


Write characterization tests
Make code testable
Refactor 1
Enable the change
Add Tests for the change
Make the tests pass
Refactor 2

It’s All About the Tests



Without tests, you cannot tell if your changes are improving the codebase, or making things worse.

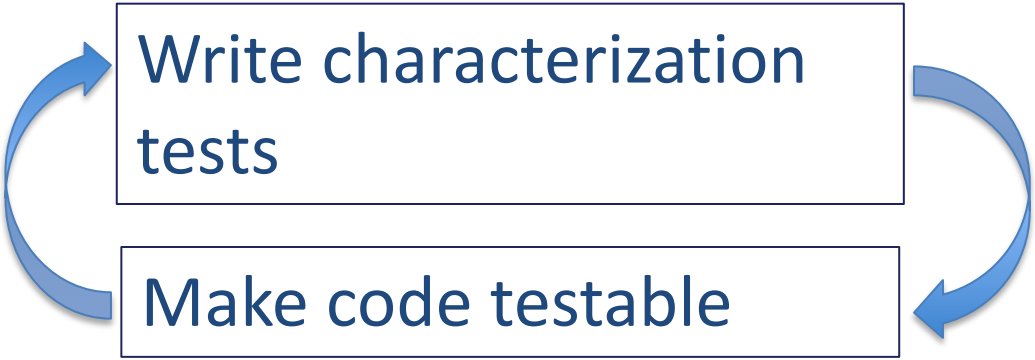
Exercise

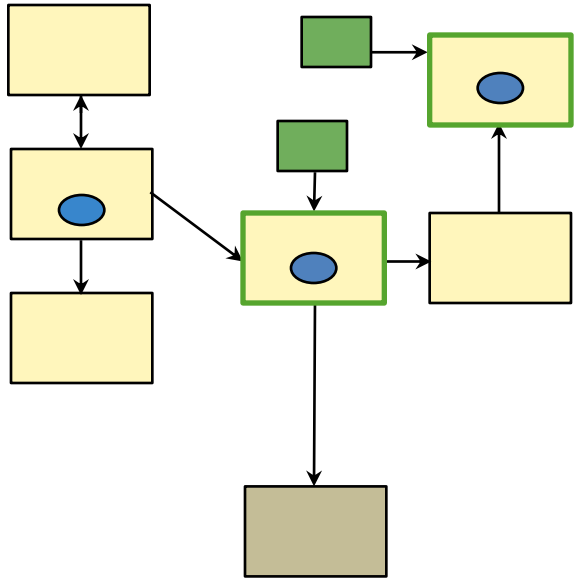
- Add characterization tests
  - Number of squares in board
  - Number of mines squares.

All tests finished?

- Can we characterize:
  - Location of number squares.
  - Number on number squares.
  - Location of mine squares.

Make Code Testable





Make Code Testable

- Make minimal changes that allow characterization tests.
  - Loosen encapsulation
    - Private → protected → public methods
  - Add or Extract methods
  - Override methods in tests

- Why not refactor now or make big changes?

Safety first

- Prefer “Safer” smaller changes to good coding practice
- OK to allow “poorer” coding practices to enable testing

- Fix when Refactoring
  - After your safety net is in place and you can make changes with proof you are not changing behavior.

Use “Sprouts”

- for fast minimally invasive tests
  - Extract method on behavior if needed
    - Prefer using tools
  - Change private to protected as needed.
  - Make derived test class and override behavior

Tool supported Refactors.

- Extract Method.
  - Tries to convert highlighted code into a separate method
- Extract interface
- Extract Class
  - Takes methods and makes a new class.
- Introduce Parameter Method
  - Makes parameters of method into a class.

Private to protected

```
public class Foo {
  private List getData(){…}

  public void barring() {
    List data = getData();
    …
  }
}
```

- Make private method protected

```
public class Foo {
  protected List getData(){}
}
```

Override method in Test

```
@Test
public void TestBarring {
 Foo subject = new Foo {
   public List getData(){
     return Arrays.asList(new Integer[]{
              32,33,34,42,44,53,37,46,64,88
              });
   }
 subject.barring();

 assertTrue( … );  // assert results based on getData()

}
```

Exercise

- Add tests that characterize
  - Location of number squares.
  - Number on number squares.
  - Location of mine squares.
  - Location of empty squares
- A sample is sometimes enough.
  - No need to check all 100 squares.
  - What are testy mine locations?

Bonus: Does current code support only 10 mines?

Write test with 20 mines

Write test with 100 mines

Write test with 0 mines

NOTE: Even if these tests generate errors, these are still characterization tests.
Make them passing tests.

Enough Tests?

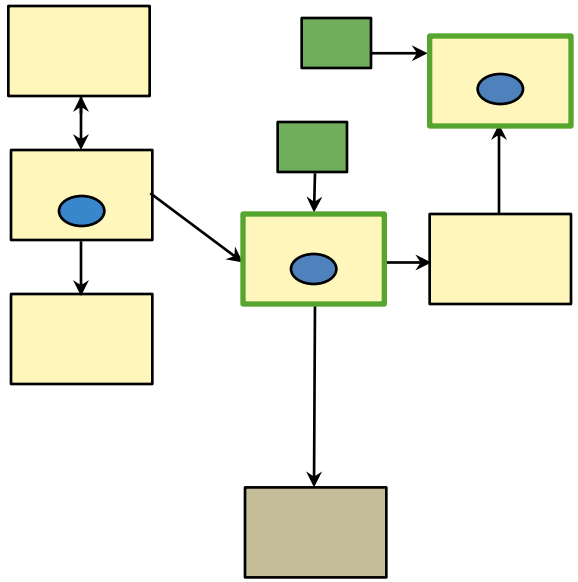
- Check your coverage.
  - What other tests do we need?
  - When it is “good enough” then proceed.

Enable changes

Write characterization tests

Make code testable

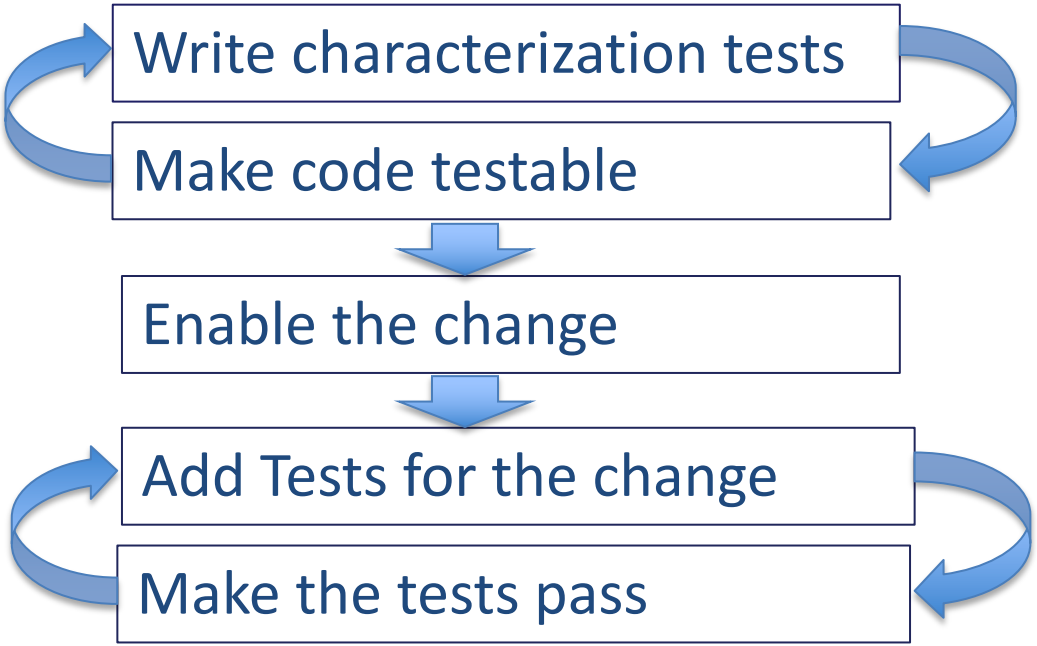
Enable the change

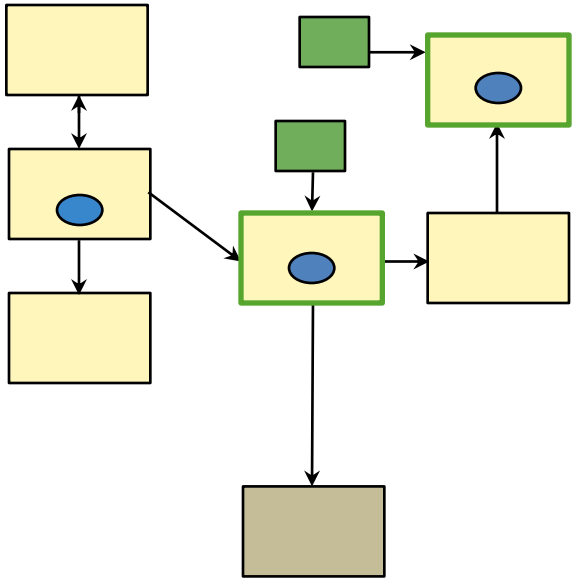


Exercise: Enable the change

- Add a test that passes a minefield height, minefield width, and number of mines into the Minefield constructor.
- Use only same values for the height, width and number of mines as the original code.
- Make the test pass.
- All characterization tests should continue to pass.

Make the change

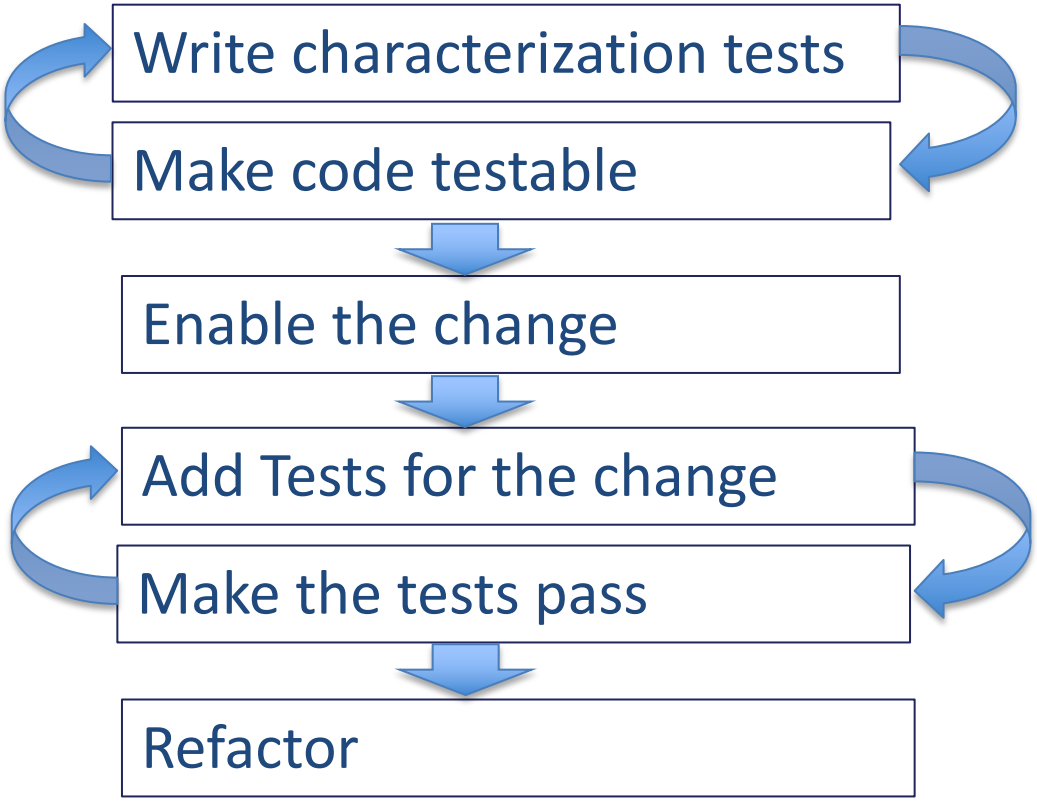


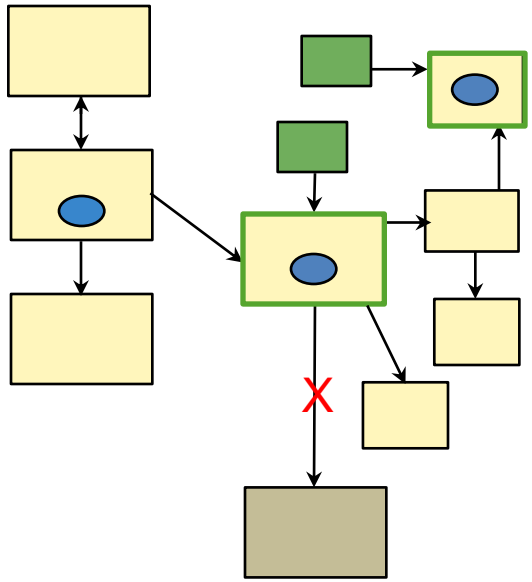


Exercise: Make the Change

- Add tests that change the minefield's
  - height
  - width
  - number of mines
- Make tests pass.

Refactor and Remove Dependencies





Make the code better

- Refactor
  - Make the code clean
  - Make the tests clean
  - Break dependencies
  - Remove temporary “test enabling” code
  - Add more tests

Sprouts are fast

- Sprouts are fast
  - They do not break dependencies
  - They are not making the code better.
- Good for getting system under test
  - Prefer to eliminate them when refactoring.

Removing Dependencies

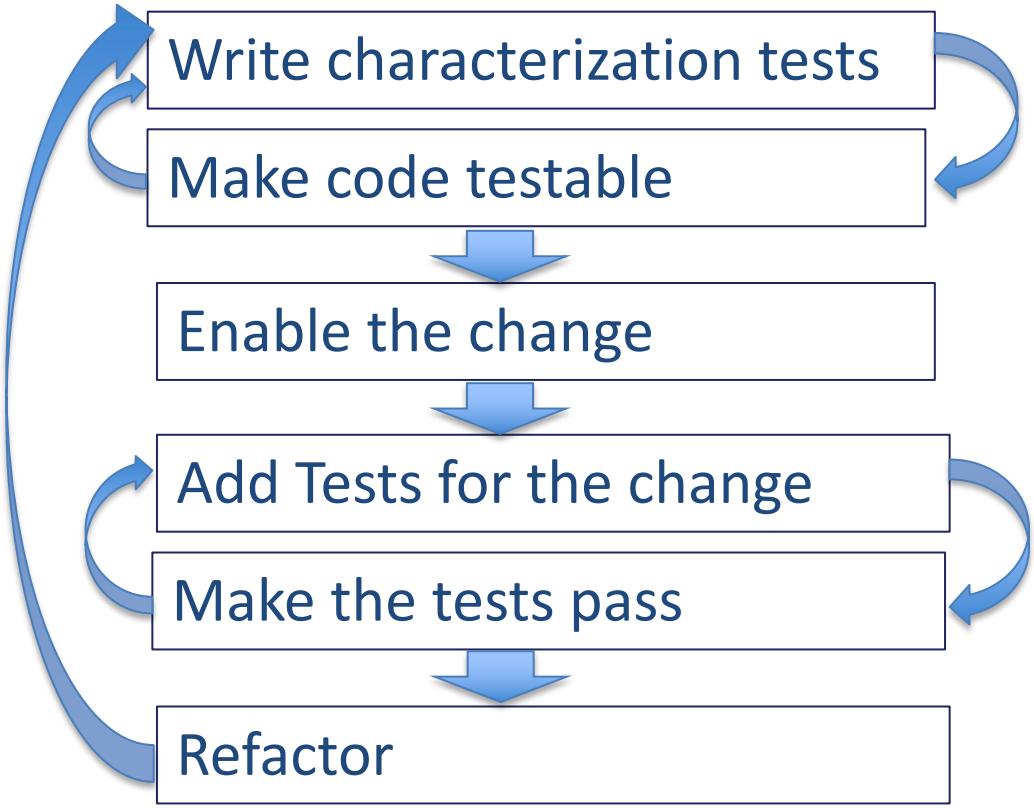
- Promote Instance Creation
  - Pass in objects instead of instantiating variables
  - Convert Functions to classes, then pass in an instance of the class.

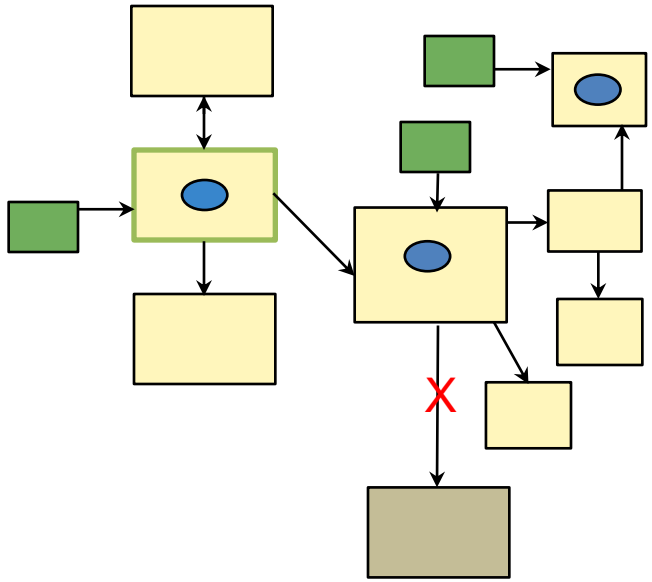
- Also called Dependency Inversion

Exercise

- Create a class that does the random number generation.
- Change the code to use the random number generation class
  - No old test should fail
  - keep the old number generation tests working.
    - Implies keeping the number generation method temporarily.
- When all tests old and new are passing, then remove the old “sprout” tests and refactor the code to remove the number generation method.

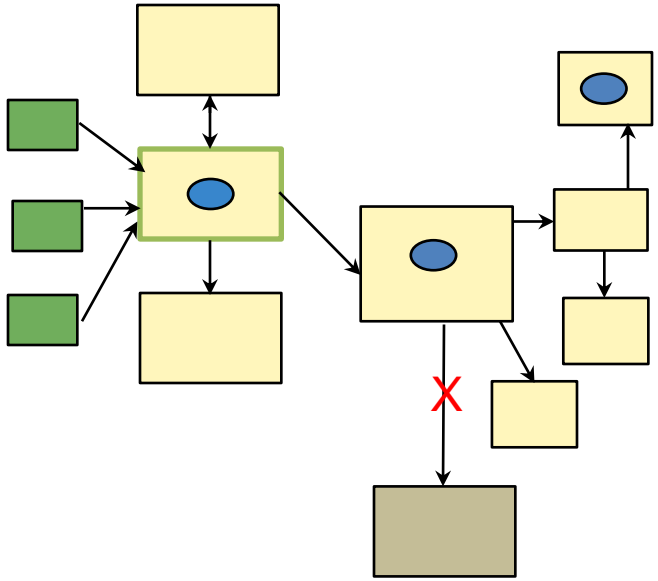
Repeat with next change point.





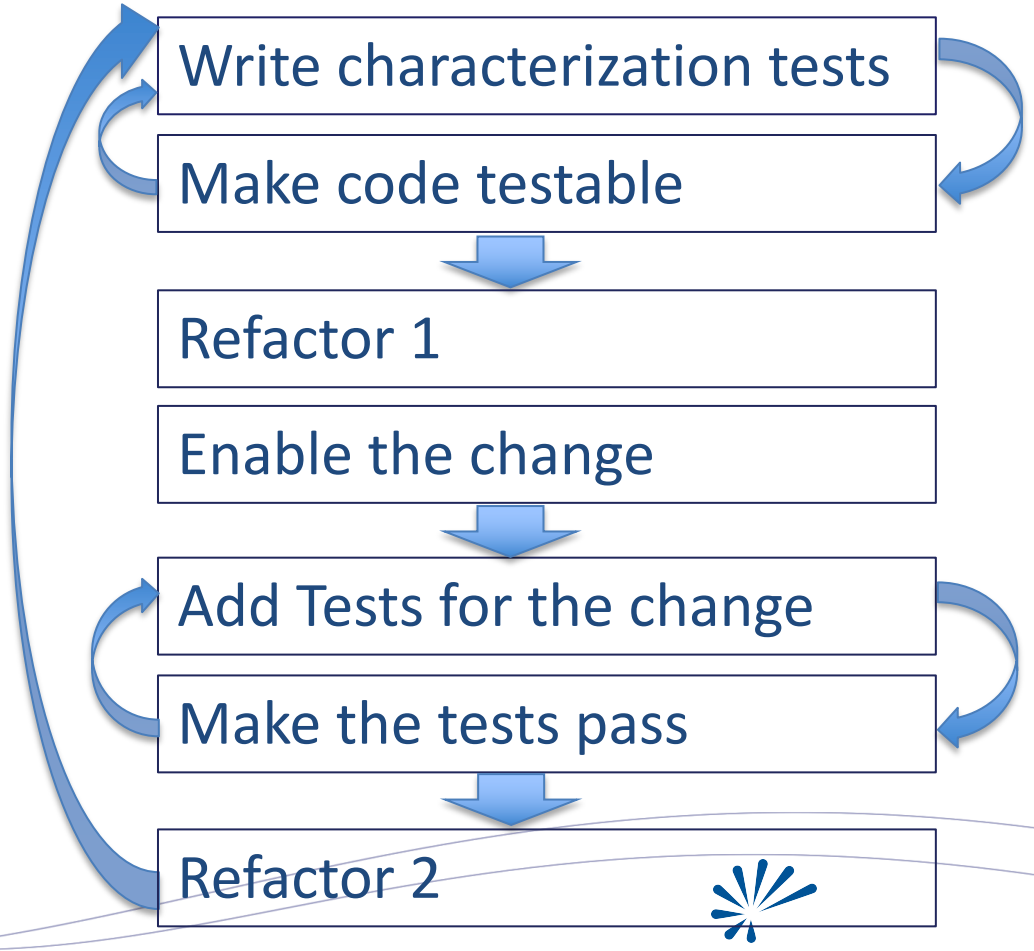
Refactor Tests

- Move tests higher in the object model
- Move towards testing behavior
  - many classes being tested together
  - Prefer testing real code
  - Avoid test doubles and mocks except for external dependencies
- Make code refactoring easier



Refactor after Characterization

- Refactor Early.
- Consider the change.
- If the current design will not accommodate the change, refactor it so that it will.

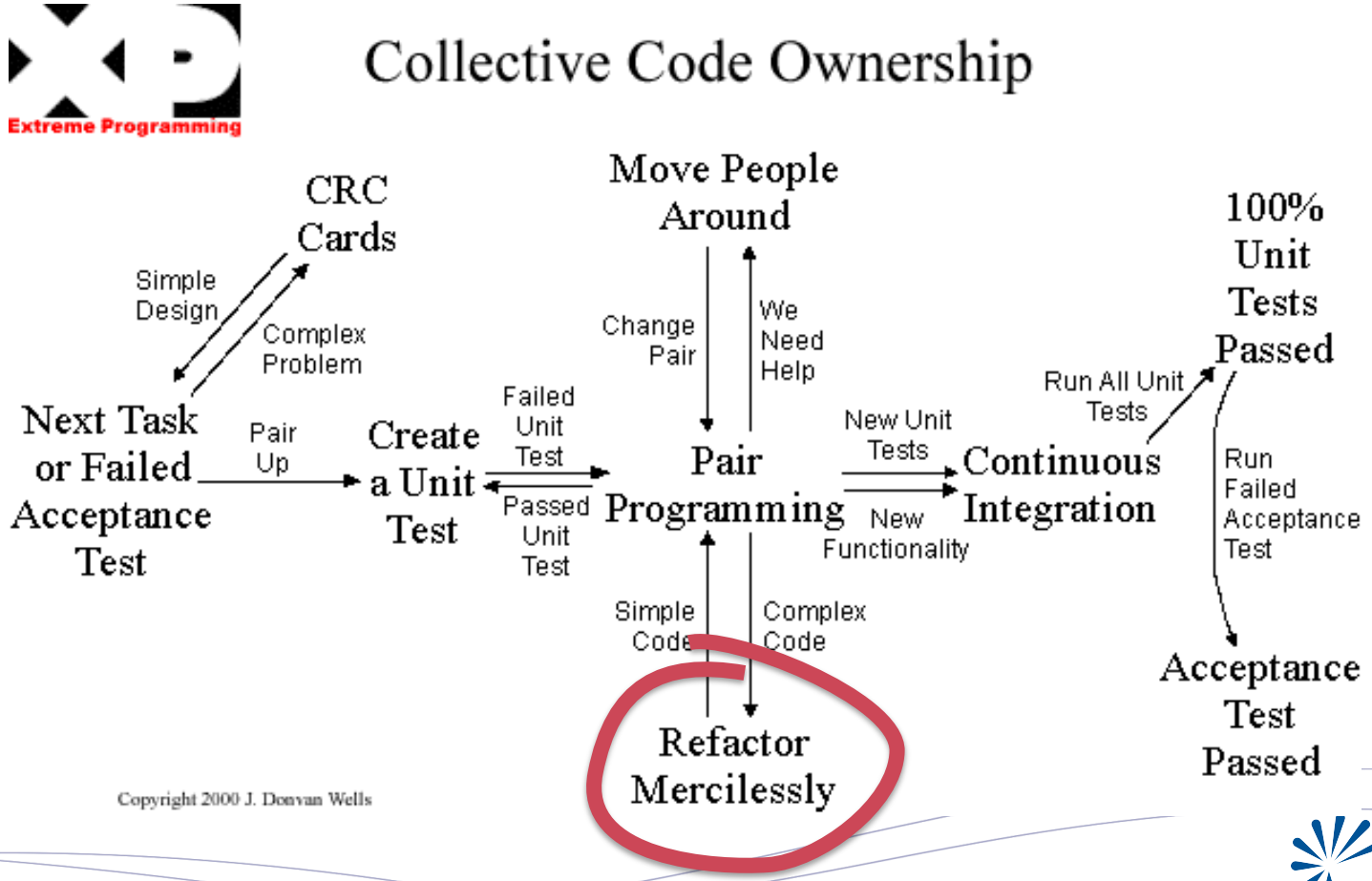


improving
It's what we do. ™

Module 10:

MAINTAINING TESTS

XP – Code Level



Keeping your Acceptance tests Sweet

- Acceptance tests require refactoring too.
  - Flickering Tests
  - Brittle Features
  - Slow Features
  - Bored Stakeholders
  - Shared Data

Flickering Tests

- Sometimes they pass and sometimes they don’t.
  - Team can’t trust the tests anymore.
  - Hard but vital to fix

Flickering tests causes

- Leaking State
  - Some state is not reset between scenarios.
- Race Conditions
  - External when finishes after the then.
- Shared environments
  - Simultaneous test runs change each others data

Brittle Features

- Changing one test breaks another
  - Tests need refactoring too
  - Abstract dependencies

Slow Features

- Takes too long to run the tests.
  - Remove dependencies

Shared Data

- Access central test data
  - Within a transaction
  - Reset the data when done
  - Use test mothers

Bored Stakeholders

- Don’t want to participate in creating examples.
  - too complicated
  - too many Incidental details
  - wrong language – use the business’s language
  - Too concrete, make more abstract.

improving
It's what we do. ™

Thank You!

improving.com


@improving