ThoughtWorks®
The art of heavy lifting.℠

Agile Methods, Testing and Quality Assurance

bret@pettichord.com
www.pettichord.com
www.thoughtworks.com

March 2005, Agile India, Bangalore

How to Invent an Airplane

How to Invent an Airplane

- Wright brothers had to learn how to test their design elements first
  - Propeller shape
  - Wing design
  - Control surfaces
- Invented the wind tunnel and used it to test propeller and wing designs
- Created and tested kites and gliders to test wing designs and control surfaces
- Had to invent the science of aeronautics
- Had to build testing into their design process

But Software isn't as Hard as Aeronautics

- Working software can actually be built using code and fix

- But like the Wright brothers,
  agile developers…
  - Test their components as part of the design process
  - Have had to create test harnesses and testing techniques
  - Make frequent **reality checks** instead of depending on the wisdom of the plan

What is Agile Development?

- Work is divided into chunks of business value
  - These ‘stories’ seem valuable to a business user
  - Customer can measure progress in their own terms
  - Requires delivery of vertical rather than horizontal collections of code
  - Customer-perspective acceptance tests determine completeness
- Work is scheduled in time-boxed iterations
  - Overhang is rescheduled for a future iteration
  - Ensures regular deliveries of working code
  - Allows team velocity to be measured
  - Contrast with RUP and other spiral methods that use scope-boxed iterations
- Team approach
  - The team contains all the necessary skills
  - The team as a whole is responsible for the success and quality of the software
  - Frequent collaboration, pairing, changing pairs and dispersed knowledge
  - Collective code ownership (optimistic locking)
- Developers write automated unit tests
  - Writing tests is seen as part of the coding job
  - Expected to run tests often
  - Breaking unit tests is always a showstopper

Responding to Change vs. Following a Plan

- Change includes learning
  - We don't know everything at the start
- Two Approaches to Planning
  - Planning is hard, therefore we must get better at it
  - Planning is hard, therefore we must reduce the need for it

- Agile development is an empirical practice focused on working code (working code *over* detailed documentation)
- The uncertainties of planning are mitigated with frequent ***reality checks***
- The biggest innovations in testing today are coming from the agile community

Reality Check: Unit Testing

- Units are functions, methods or classes.
- Unit tests are in the same language as the code being tested.
- Unit tests are written by the programmers who wrote the the code being tested.
- A test harness or framework collects tests into suites and allows them to be run as a batch.
- The X-Unit frameworks are popular harnesses.
  - JUnit for Java, NUnit for Dot-Net…
- Most agile developers are ‘test-infected’

Types of Unit Testing

Unit isolation testing *Test each unit in isolation*	*Create stubs for external units*	•Use Mock Object classes
Unit integration testing *Test units in context*	*Call external units*	•Introduces dependencies. •Test suites take longer to run

- Many agile developers strongly prefer unit isolation tests: “true unit tests”
  - Run faster, therefore run more often
  - Less likely to break when refactoring other code

Refactoring
Improving the Design of Existing Code

- Refactoring restructures code (hopefully for the better) without changing its behavior.
- Unit tests define behavior and therefore determine whether behavior was inadvertently changed.
- Traditionally, lack of unit tests have discouraged developers from refactoring, resulting in brittle code.

- *Refactoring*, by Martin Fowler
  - Testing is an integral component to refactoring.
  - 9 of the 17 “sound bites” mention testing.

Test-Driven Development

- Developers write unit tests before coding.
  - Motivates coding
  - Improves design
    - *reducing coupling*
    - *improving cohesion*
  - Provides regression tests
- An approach to design
  - More than just as test strategy
  - Specification by Example
  - Focuses programmer on how callers will use the code
  - Spawning new lightweight frameworks using dependency injection.

```
public void testMultiplication() {
  Dollar five = Money.dollar(5);
  assertEqual(new Dollar(10), five.times(2));
  assertEqual(new Dollar(15), five.times(3));
}
```

Test-Driven Development: Red-Green-Refactor

1. Write a test, then run it. Make sure it fails. RED
2. Make the test pass. GREEN
   - Use the simplest design that will work.
   - Bad design (duplication, etc) is OK! (for now)
   - Add code only when tests demand it.
3. *REFACTOR* to improve the design.
   - Now, remove duplication
   - Unit tests are the ***reality check*** to let you know you didn't break anything

Reality Check: Continuous Integration

- Rebuild the code whenever a new commit is made
- Then run the unit tests
- Post results to the web
- Send email with any errors

- Tools:
  - Cruise Control
  - Damage Control

Reality Check: Spikes

- The agile approach to architecture
- A spike is throwaway code that explores a particular approach to assembling code
  - Will it work?
  - How will it perform?
  - Is it ugly?

Reality Check: Frequent Delivery of Business Value

- Not just a hunk of code
- Actual functionality that is valuable to end users
- A vertical rather than a horizontal slice
- Delivered to the customer
- Allows customer satisfaction to be measured
- Regular ‘Beta’ testing throughout development
- Usability testing, exploratory testing
- System must remain stable to make this happen (hence the need for automated regression tests)
- Focus on true customer satisfaction rather than just meeting the letter of the requirements

Reality Check: Immediate Acceptance Testing

- A story isn't done until it has been tested
- Usually tested in the iteration
- "Sometimes you just have to throw a turkey in the engine."

Automating Acceptance Testing

- Characteristics of Successful Test Automation Projects…
  - Collaboration between testers and developers
  - Automate early
  - Team commitment (vs “it would be good if”)
- Agile teams have all three

- Agile Testing Rules
  - Programmers write automated unit tests.
  - Acceptance tests must also be automated.
  - Programmers and testers work together on acceptance tests.

Challenge: Regression Test Tools

- Most commercial test tools work poorly in an agile environment. Most have these flaws:
  - Vendor-specific languages (vendorscripts)
  - Poor integration with source control
  - Hard to use with continuous integration
  - Impractical to install on every workstation
- These problems make them impractical for use by the team as a whole.
- Agile teams are building their own test tools and releasing many of them as open-source…

Problems with Commercial Test Tools

- Proprietary Scripting Languages
  - *Winrunner (TSL), SilkTest (4test), Robot (Test Basic)*
  - *http://www.stickyminds.com/se/S2326.asp*
  - But newer tools are now using standard languages
    - *Astra QuickTest (VB Script), XDE Tester (Java),*
- Incompatibility with Source Control
  - Temporary files and directories (WinRunner)
    - *http://paulhammant.com/blog/000245.html*
  - Key information stored in repositories (Rational)
- Lack of External Calling API's
  - They refuse to allow themselves to be used as a library.
  - Generally, you can only launch complete scripts with limited access to results information.
  - Therefore difficult to integrate with Continuous Integration
  - Some new low-cost and shareware tools are exceptions
    - *E.g. TestComplete*
- Restrictive and Expensive Licensing
  - Developers can't run test suites.

These "features" encourage vendor-lock and frustrate serious programming

- Open-Source Tools almost always avoid these shortcomings.

Watir

- Watir is a Ruby-library that drives the IE browser.
  - Bret Pettichord & Paul Rogers
- Website
  - http://wtr.rubyforge.org
- Mailing List
  - http://rubyforge.org/projects/wtr/

Selenium

- Selenium is server-side software that delivers a JavaScript browser-bot that runs inside IE, Firefox or Mozilla.
  - Jason Huggins & ThoughtWorks
- Website
  - http://selenium.thoughtworks.com

QA Paradigm #1: Quality Assurance is Testing

- Most QA people are actually employed as testers
- “Did you QA this?”
- “Independent testing is better testing”
- Are used to testing untested code and struggle when working with agile developers
  - E.g., overuse of boundary testing is common

QA Paradigm #2: Quality Assurance is Process

- Role defined by CMM and IEEE
- An approach that many QA groups aspire to
- The Process Police must force discipline on the developers
- However, test teams that also try to enforce process may undermine their effectiveness as testers
  - discourages communication
  - reduces trust
  - may cause delays
- Also, tend to enforce waterfallian practices, which is counterproductive for agile teams

QA Paradigm #3: Quality Assurance is Team Responsibility for Customer Satisfaction

- "Whole Team" means that QA can't be delegated to a person or subgroup
- Everyone is responsible for raising quality issues
- It's not enough to say that you did what they asked for
- Quality ultimately is defined by the customer, not by process standards, nor by stale documents
- This is the approach preferred by Agile teams

Agile Is About **Reality Checks**

- This conference is a chance for you to make another reality check.
- Agile is not about doing what the experts say.
- It is about doing what works.
- Ask the speakers how agile methods have or haven't worked for them.

Open-Source Test Tools from ThoughtWorks

Dashboard
http://dashboard.sourceforge.net/
hloader
http://hloader.sourceforge.net/
jfcUnit
http://jfcunit.sourceforge.net/
MockMaker
http://mockmaker.sourceforge.net/
NMock
http://opensource.thoughtworks.com/projects/nmock.jsp
Marathon
http://marathonman.sourceforge.net/
Marathon.NET
http://marathonnet.sourceforge.net/
PyUnit
http://opensource.thoughtworks.com/projects/pyunit.jsp
SelfEsteem
http://selfesteem.sourceforge.net/
XMLUnit
http://xmlunit.sourceforge.net/

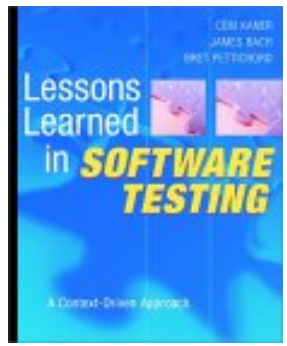
Unit Testing References

- Code First
  - Pragmatic Unit Testing: In Java with JUnit, Hunt & Thomas
  - "Learning to Love Unit Testing," Thomas & Hunt
    - http://www.pragmaticprogrammer.com/articles/stge-01-2002.pdf
  - "JUnit Test Infected: Programmers Love Writing Tests," Gamma & Beck
    - http://junit.sourceforge.net/doc/testinfected/testing.htm
  - "JUnit: A Cook's Tour," Beck & Gamma
    - http://junit.sourceforge.net/doc/cookstour/cookstour.htm
  - "Simple Smalltalk Testing: With Patterns," Kent Beck
    - http://www.xprogramming.com/testfram.htm
- Test First
  - Test-Driven Development: A Practical Guide, David Astels
  - JUnit Recipes, J.B. Rainsberger
  - Unit Testing in Java: How Tests Drive the Code, Johannes Link
  - Test-Driven Development: By Example, Kent Beck

Further Study

Context-Driven Testing

- *Lessons Learned in Software Testing: A Context-Driven Approach*
  - Cem Kaner, James Bach & Bret Pettichord
- Mailing List
  - http://groups.yahoo.com/group/software-testing/
- Wiki
  - http://www.context-driven-testing.com/wiki/



Agile Testing

- Agile Testing Papers
  - http://www.testing.com/agile
- "Where are the Testers in XP?"
  - http://www.stickyminds.com/s.asp?F=S6217_COL_2
- Mailing List
  - http://groups.yahoo.com/group/agile-testing/

Open Source Test Tools

- Home Brew Test Automation
  - http://www.io.com/~wazmo/papers/homebrew_test_automation_200409.pdf