Agile Development with XP

Ian McFarland, Principal
Pivotal Computer Systems

Who are we?

- An agile development consultancy
- Java, Ruby on Rails, AJAX/Web 2.0
- Co-development and Coaching
- Fishing and teaching to fish

Who are you?

Agile Development

- A family of development methodologies* with common goals:
  - Short iterations
  - Change-friendly
  - Aligned with real customer needs

* Scrum, XP, Crystal, Context-Driven Testing, Lean Development, RUP, or anything else that fits the principles of agile

Agile Manifesto

Agile values...

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

- http://www.agilemanifesto.org/

Extreme Programming

- One of the (if not *the*) leading Agile methodologies
- A disciplined methodology with an unfortunate name
- A collection of industry best practices, turned up a few notches, and integrated into a single development methodology
- Not analogous to extreme sports

XP: What is it?

- Clear, customer visible stories
- Test-Driven Development
- Continuous Integration
- Short Iterations
- Pair Programming
- Extensive Customer Involvement

Customer-Visible Stories

- The focus on customer-visible stories keeps development concrete, and fights over-architecture
- It reminds us why we're doing the work in the first place
- It forces us to justify our design choices, grounding them in the real customer need

The Planning Game and the Point System

- Stories are estimated each week during a meeting traditionally called *The Planning Game*
- Stories are broken down into units of 1, 2 or 3 points
- Points measure complexity, not duration
- Larger stories are broken down into smaller stories that are 3 points' worth or smaller
- The customer prioritizes the work, informed by the complexity estimates

Task Estimating

- People are better at estimating complexity than duration
- Tasks have fractal complexity: Small tasks are more predictable than large ones
- Exposing the cost of features and giving control to the customer creates alignment between the developer and the customer

The Point System

- One point: I know exactly how to do this, and can do it in half a day.
- Two points: I know exactly how to do this, but it will be some work.
- Three points: "Somehow we will implement this feature."
- Three points almost always turns into more, and is a big red flag that the story needs to be broken down into smaller stories.

Velocity Tracking

- A focused team gets about as much done each week as it did the week before
- The number of points completed in one week is an excellent predictor of the number of points completed in the next
- Predictable results build trust between developer and customer

Test-Driven Development

- You write the tests before you write the code
- Separates requirements from implementation
- Produces an executable specification, and documentation that stays in sync with the code
- No code without a test
- 101% test coverage (well, at least at the start.)

Why do we care so much about tests?

- The obvious reasons, but they're secondary

Why do we care so much about tests?

- The obvious reasons, but they're secondary

The real reasons:

- Separating requirements from implementation frees you from the tyranny of preoptimization
- Driving from tests forces modular, usable design
- Complete test coverage lets you refactor with impunity

Red-Green-Refactor

- Write a test that expresses what your code is supposed to do (and that fails)
  - ...and often you'll write several layers of failing test before you write a line of code
- Write code that makes the tests pass
- Refactor the code to improve design, reduce duplication, improve code clarity
- Make sure the tests still pass when you're done
- Check in

Continuous Integration

- Everyone runs the entire test suite before submitting
- The continuous build checks out the trunk, builds it, and runs all tests
- Problems are caught early, when they're easy to fix
- The entire application is kept in a deployable state from the first week

Short Iterations

- One week iterations make for easier course corrections, and shorten the feedback cycle
- Requirements change as the customer has a chance to validate the design through play testing
- Complete test coverage and alignment between customer and developer make course corrections painless instead of arduous, cheap instead of expensive

Pair Programming

- Probably the most controversial part of XP
- How can it be faster for two developers to work on the same problem? Surely it's faster if they work on two separate problems in parallel...
- Yes, we actually work this way, with two developers working on a single machine.

We do this naturally

- Developers will instinctively 'pair program' when one introduces another to a new code base
- Developers will instinctively work together to solve hard design problems

  But...

- Developers don't always want to be so closely scrutinized.

How Pairing Works

- Pairing accelerates knowledge transfer
- Pairing makes deep problems shallow
  - Your 80/20 rule overlaps favorably with your partner's 80/20 rule
- Pairing keeps you focused
- Pairing keeps you honest

How do you write code?

- Ask yourself what percentage of your development time you spend...
  - ...stuck on some difficult problem
  - ...stuck on some trivial problem
  - ...trying to choose between two implementation choices
  - ...reading email or news or blog posts

Customer Involvement

- The customer owns the priorities, the developer owns the cost estimates
- A feature isn't done until it's deployed to the demo server *and* approved by the customer

  The results:

  - Features are really done when they're marked done
  - The customer and developer are aligned in reaching their common goal

The Agile Rhythm

- The Planning Game
- The daily stand-up
- Red-Green-Refactor (sync-green-submit)
- Deployment to demo and customer approval

Q&A

- Pivotal offers a three week apprenticeship program on agile development using Ruby on Rails from our San Francisco office

- For more information, contact me.

Ian McFarland <ian@pivotalsf.com>