Agile from the Front Lines

Fred George

ThoughtWorks

fred@ThoughtWorks.com

March 4, 2005

Agenda

- Foundation Process: J-I-T
- Lessons from the Front Lines
  - And keys to failure
- Uber-Extreme Programming
- Discussion

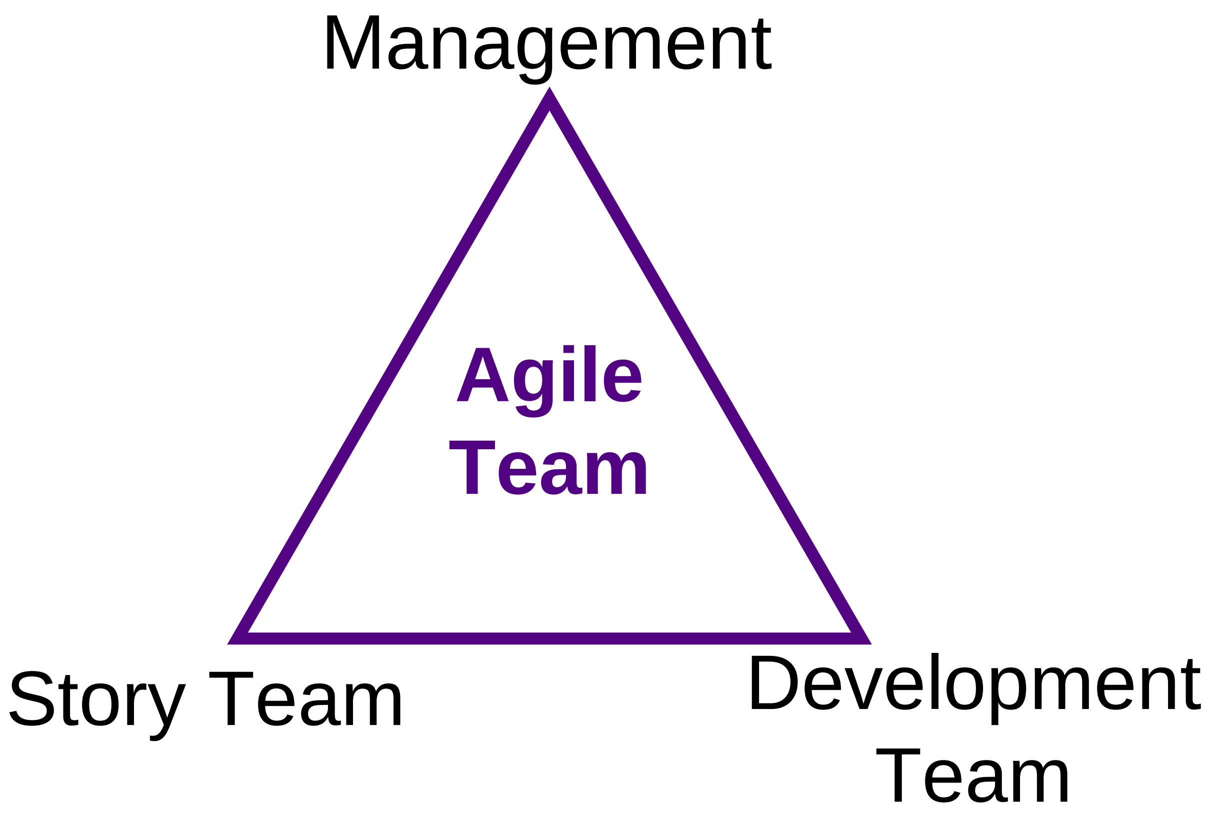
Foundation Process – J-I-T

- Just-in-Time is a Manufacturing Technology
  - Originally Japanese auto manufacturing
  - Focused on cost control
  - Quality resulted from fast feedback cycles
  - Now also called “**Lean**”
- XP is “Software Manufacturing”
  - Focus on continuous delivery of software
  - Line can be sped up or slowed down
  - Innovation is required for each piece produced

XP Guidance From J-I-T

- Feedback Cycles Identify Defects
- Aggressive Definition of “Waste”
  - Anyone who is not producing (code, stories, tests)
  - Any artifact not required for production
  - Over-testing
- Reduction in Staff Roles
- Reduction of Specialists

Agile Triangle



Selling Agile to:

- Programmers:
  - Continuous delivery
  - Utilization of all their skills
  - Fun
- Executives:
  - Financials
  - Time to market

Waterfall vs. Agile*

- Waterfall
  - $28 / hour
  - 12 months
  - $2M USD

- Agile
  - $40 / $80 / $130
  - 8 months
    - 3 releases
    - 3 months to first
  - ***$1.1M USD***

***Consultants Report at www.ThoughtWorks.com**

It’s the Stories, Stupid

- Delivery is What the Customer Wants
- Not the Process
- Not the Technology
- Aligns with ***Lean*** Management

Count What Counts

1. Stories
2. See #1
3. Report on:
   - Test count – measure of function
   - Automated acceptance test coverage
   - Class counts
4. Never Count:
   - Task cards
   - Lines of code

Harvest Legacy Artifacts

- Requirements Already Written?
  - Exploit and convert to Stories
- DB Design in Place?
  - Great! Information needs in place.
- Developers of the Legacy System
- Domain Experience (QA, Development)
- Legacy Code (with new tests)

Delivery-Focused Stand-Up

- Stand-up Meeting Reports
  - What I did yesterday…
  - And what I will do today
- Not a Design Session
- Assign Actions and Move On

“I didn’t finish my task”

- Break Task Down if Complex
  - Bring back more tasks
- Re-assign Teams
- Don’t Blame – Just Fix

Which Practices Do I Need?

- "I don't know enough (yet) to discard any of them."
- Biggest Early Mistake: Discarding a Practice
  - Difficult to add in later
  - Risk is incurred, without understanding it
- Iteration Meetings Provide Forums
  - Process feedback is essential to iterations
  - Provides forum for discussion and change

Most Discussed Practice?

- Pair Programming
- Managers Should Love It
  - Provides cross training
  - Allows easy introduction of new members
  - Easy to exploit (and harvest) drop-in specialists
- Dissertation Shows Productivity Sound
  - With big boost in quality
- Control vs. Smart Keyboard

Most Difficult Practice?

- Simple Design
  - Best design for all function delivered
  - Little regard for tomorrow’s requirements
  - YAGNI
- Steps (in order!)
  - Works
  - Communicates
  - No duplicate code
  - Fewest classes and methods

Most Neglected Assumption?

- XP Depends on OO Programming
- XP Assumptions:
  - Design for today, not tomorrow
    - *…because it is easy to change*
  - Refactoring
    - *…using object patterns*
- Do OO differently
  - Legacy system: 79 classes
  - Replacement system: 1400 classes

Roles vs. Jobs

- Many Roles Exist
  - Agile developer can play many roles
  - The more roles, the more valuable
- “I’m an architect”
  - “So, you can draw pretty pictures?”
- “Coach” is a Role, not a Job
  - Coaches can’t play
  - “Player-Coach” is a job

Judging Contributions

- Q: How Do You Measure Individuals?
  - Not by counting lines of code
  - Not by counting task cards
- A: Ask the Team
  - Who do you like to pair with?
  - Who is helping out others?
- No hiding for poor programmers

Fight the Bad Press

- “Extreme anything” Scares Management
- XP is Rigorous
  - Not “cowboy coding”
  - 12 practices well-defined and very demanding
- Not an Excuse for “No Documentation”
  - Documentation is a valid ***Communication*** mechanism

Uber-Extreme Programming

- Smaller Task Cards
  - 2-4 hours each
  - Constant check-in races
- No formal re-estimating during iteration
  - … and who will use this information?

Uber-Extreme Programming

- Collective Code Ownership => Collective Requirements Ownership
  - Tasks don’t have pre-assigned owners
  - Invites unwanted, time-consuming questions
- J-I-T Pairing
  - Appropriate pair for most important task at that moment
  - Pairs not self-selecting
    - Match skills to tasks
    - Meet secondary objectives

Discussion

- What Lessons Have You Learned?
- Had Success Doing “Partial XP”?