What Is This Talk About?

- Focus on Extreme Programming
- This is a tutorial presentation
- Learning XP is somewhat experiential
- Time for questions and discussions
- Explanation of how XP is intended to work
- Not trying to evangelize or defend XP
- Debates can follow, time permitting
What Are Your Impressions of XP?
Who Is Your Presenter?

Your presenter is Paul Hodgetts

- Founder and CEO of Agile Logic, based in Southern California
- Agile Logic provides training, coaching, consulting, and custom development services
- 20+ years experience in all aspects of software development
- Served as coach, developer, and mentor for agile teams for the past 4+ years
- Recent focus on agile processes, enterprise Java, and organizational change management
- Contributing author to Extreme Programming Perspectives
- Previous presenter at XP/Agile Universe and JavaOne
- Taught at C.S.U.F, member of XP, Java, J2EE, C++ advisory boards
- Contact info: www.AgileLogic.com phodgetts@agilelogic.com
Agenda

- Agility
- History of Extreme Programming
- XP and the World of Agile Processes
- XP Overview
- XP Project Community
- XP Process Cycle
- XP Practices
- Other XP Topics
What Do We Mean By “Agile?”

According to the Merriam-Webster on-line dictionary “agile” means:

“1: marked by ready ability to move with quick easy grace;”

“2: having a quick resourceful and adaptable character.”

In agile software development, “agile” tends to mean “the ability to respond to change.”
Change In Projects

- Changes From Requirements
  - Customers Learn from the Solution
  - Business Environment and Conditions Change
  - Business Processes are Re-engineered

- Changes From Technology
  - Tools/Platform Release New Versions
  - Actual Tool/Platform Capabilities May Vary from Plans

- Changes From People
  - Interactions are Complex
  - Individual Behavior is Unpredictable
What’s Really Different About “Agile?”

- **“Ad-Hoc” Processes**
  - Still the Most Common Process
  - Not Necessarily Chaotic, Just Not Consistent

- **“Defined” Processes**
  - Creation of Comprehensive Activity-Based Plans
  - Execution of Defined Activities
  - Management by Controlling Activities to Conform to Plan

- **“Adaptive” Processes**
  - Setting of Goal-Based Objectives for Deliverables
  - Opportunistic, Local Planning and Execution of Activities
  - Management by Retrospection, Learning, Adaptation
Characteristics of Agility

- Empowered, self-organizing teams
- Multi-discipline, cross-functional teams (whole team culture)
- Project- and product-centric focus, minimal organizational focus
- Shared responsibility, role-based accountability
- Shared vision of standards of excellence
- Close, continuous collaboration, direct communication
Characteristics of Agility

- Early, frequent, and continuous demonstration of progress through concrete deliverables
- Rapid feedback, reflection, learning, adjustment
- Small work batch sizes, minimal specialization, reduced queuing delays
- Just in time production, minimize production of artifacts not immediately (or ever) consumed
- Low friction – simplicity, minimalism, pragmatism
- Avoidance of “debt,” focus on forward movement
- Parallelism and opportunistic control
- Sustainable, constant, predictable pace
Goals & Potential Benefits of Agility

- Delivering the most value to the business, efficient use of resources, maximize ROI and time-to-ROI
- Faster development, higher productivity
- Flexibility to respond to change and leverage learning
- Better quality solutions, more enduring systems
- More fulfilling development culture
Agenda

- Agility
- History of Extreme Programming
- XP and the World of Agile Processes
- XP Overview
- XP Project Community
- XP Process Cycle
- XP Practices
- Other XP Topics
History of Extreme Programming

- Early Influences
  - Incremental, stakeholder-driven design process from Alexander
  - Programming as learning from Papert, Kay
- Kent Beck & Ward Cunningham
  - Mid-80s – Pair programming at Tektronix
  - 80s, 90s – Smalltalk culture produces refactoring, continuous integration, constant testing, close customer involvement
  - Generalized to other environments
  - Early 90s – Core values developed within patterns community, Hillside Group
History of Extreme Programming

- 1995 – Kent summarizes in Smalltalk Best Practices
- 1996 – Ward summarizes in Episodes
- 1996 – Kent adds unit testing, metaphor at Hewitt
- 1996 – Kent takes on Chrysler C3 project
- C3 adds Ron Jeffries as coach
- Practices refined on C3, published on Wiki
History of Extreme Programming

- Scrum practices incorporated and adapted as planning game
- 1999 – Extreme Programming Explained
- 1999 – Fowler publishes Refactoring
- 1999 – XP Immersion held, e-group formed
- 2000 – more books, first conferences
- Evolution continues through today
Agenda

- Agility
- History of Extreme Programming
- **XP and the World of Agile Processes**
  - XP Overview
  - XP Project Community
  - XP Process Cycle
  - XP Practices
- Other XP Topics
Agile Processes

- Extreme Programming
- Scrum
- Lean Development
- DSDM
- Crystal
- Adaptive Software Development
- Feature-Driven Development
Agile Alliance

- 2001 – representatives from agile processes meet in Snowbird, Utah
- Agreed on a “manifesto” of values and principles
  - Individuals and interactions over processes and tools
  - Working software over comprehensive documentation
  - Customer collaboration over contract negotiation
  - Responding to change over following a plan
- “That is, while there value in the items on the right, we value the items on the left more.”
Agenda

- Agility
- History of Extreme Programming
- XP and the World of Agile Processes
- **XP Overview**
- XP Project Community
- XP Process Cycle
- XP Practices
- Other XP Topics
What Is Extreme Programming?

- XP is a specific instantiation of an agile process
- XP combines best practices in a different way
- XP is a different approach to development
- XP provides a core process model
- XP is not intended to be a complete framework
Emergence

- XP provides values and principles to guide team behavior
- Team expected to self-organize
- XP provides specific core practices
- Each practice is simple and self-complete
- Combination of practices produces more complex emergent behavior
- Synergy of practices still not fully understood
Key Assumption of XP

- The cost of change curve can be flattened
- How?
  - Better program organization (object-oriented)
  - Simple designs kept simple with refactoring
  - Automated testing to maintain integrity
  - Better tools and faster machines
- Because of this we can:
  - Make decisions later, defer costs, keep options open
  - Reduce the time-to-ROI with partial solutions
  - Learn from feedback, adapt to change
Why Is It Called “Extreme?”

- Selected the minimal set of effective practices
- “Turned the knob up to 10” on each practice
  - Very short cycles (planning game)
  - Continuous code reviews (pair programming)
  - Extensive testing (unit testing, acceptance testing)
  - Continuous integration
  - Constant design improvement (refactoring)
  - Continuous architecture refinement (metaphor)
  - Etc…
XP Values

- Communication
- Simplicity
- Feedback
- Courage
XP Principles

- Rapid Feedback
- Assume Simplicity
- Incremental Change
- Embracing Change
- Quality Work
- Teach Learning
- Small Initial Investment
- Play to Win

- Concrete Experiments
- Open Honest Communication
- Work With Instincts
- Accepted Responsibility
- Local Adaptation
- Travel Light
- Honest Measurement
XP Project Community

- Emphasis on the “Whole Team”
- Collaboration and colocation
- Three general roles
  - Customer
  - Developer
  - Manager
- Roles define areas of accountability
- Specific job functions neither specified nor excluded (e.g., QA, PM, operations, etc.)
Customer

- Accountability for:
  - Knowing and communicating business needs
  - Definition of features (requirements, scope)
  - Prioritization
  - Test for acceptance of features
- Customer speaks as a single voice to team
- Customer could be multiple stakeholders
- Customer could be a community
- Customer is not always the principal (proxies)
Customer

Potential members of the Customer team:

- Product Managers
- Marketing, Sales
- Business Analysts
- Quality Assurance (testing aspects)
- End Users, Their Managers
- Business/System Operations
Developer

Accountability for:

- Knowing and communicating solutions
- Cost estimations and explaining trade-offs
- Delivering usable functionality that meets requirements and priorities
Developers

Potential members of the Developer team:
- Programmers
- Architects & Designers
- Technical Leads
- Interface Architects/UI Designers
- Database Designers & DBAs
- Operations & Network Designers
Manager

Accountability for:
- Defining overall organizational goals
- Interfacing with organizational entities (status)
- Environmental issues (facilities, equipment)
- Cultural issues (organizational values)
- Personnel administration (reviews, hiring, etc.)
- Business administration (budgets, etc.)
Manager

Potential members of the Manager team:
- Owners, Shareholders
- Board of Directors
- Executive Management
- Project Management
- Technical Management, Administrative Management
- Process (Quality & Process Engineering)
- Coaching
Agenda

- Agility
- History of Extreme Programming
- XP and the World of Agile Processes
- XP Overview
- XP Project Community
- **XP Process Cycle**
- XP Practices
- Other XP Topics
XP Process Cycle

- XP is iterative and incremental
- XP is driven by time-boxed cycles
- The rhythm of the XP process is crucial
Forward-Driving Activities

Each level of activity provides the \textit{minimal} materials needed for the next level

- Product activities provide materials for release cycles – requirements and priorities
- Release planning sessions provide materials for iteration cycles – prioritized & sized stories
- Iteration planning sessions provide materials for task cycles – task breakdowns
- Task development provides materials for development episodes
- Development episodes produce product
Product Life Cycles

- Involves chartering, strategy planning, feature set definition and planning, investment and resource commitments…
- Tends to be organizationally context dependent
- XP does not provide specific practices for this
- XP assumes the Customer does these things
- Primary deliverable in XP are stories
Releases

- Whole team gathers
- Retrospect and review prior progress
- Customer presents stories
- Stories are discussed (analysis)
- Developer determines technical approach & risk (architecture & design)
- Developer provides first-level estimates & options
- Customer prioritizes stories
- Customer chooses target release time box
- Stories arranged into probable iterations
- Begin the next iteration
- Primary deliverable is the release plan
Iterations

- Whole team gathers
- Retrospect and review prior progress
- Customer presents stories for the iteration
- Stories are discussed in greater detail (analysis)
- Developer determines detailed technical approach (architecture & design)
- Developer creates task list for each story
- Preliminary deliverable is the iteration plan
- Begin the development of the tasks
- Final deliverable is a deployable system
- The system should be deployed and promoted as far as possible each iteration
Tasks

- Developer signs up for a task
- Developer begins episodes to implement
- Developer ensures task is complete
- If last task, Developer ensures story is complete via acceptance tests
Development Episodes

- Developer obtains a pair partner
- Pair verifies understanding of story for this task (analysis)
- Pair determines detailed implementation approach (detailed design)
- Pair begins test-driven cycle of write test, implement to pass, refactor
- At appropriate intervals, pair integrates to code base
- Pair retrospects on progress frequently
- Pair continues until pair changes or task complete
Feedback

- Pairs are constantly communicating within themselves and outward to team.
- Daily “stand-up” meetings provide overall team status, re-synchronization, and micro-planning.
- Retrospectives provide overall status and points for process adjustment and improvement.
- Development cycles may cause rethinking of tasks.
- Task development may cause rethinking of stories.
- Story re-estimation may cause iteration changes or recovery.
- Iteration results may cause changes to release plan.
Agenda

- Agility
- History of Extreme Programming
- XP and the World of Agile Processes
- XP Overview
- XP Project Community
- XP Process Cycle
- **XP Practices**
- Other XP Topics
The Original 12 Practices

- On-Site Customer
- The Planning Game
- Small Releases
- Testing
- Simple Design
- Pair Programming
- Refactoring
- Continuous Integration
- Collective Ownership
- Coding Standards
- Metaphor
- 40-Hour Week
Evolving Practices

- On-Site Customer
  - Whole Team
- The Planning Game
  - Release Planning
  - Iteration Planning
- Testing
  - Acceptance Testing
  - Unit Testing
  - Test-Driven Development
- Refactoring
  - Design Improvement
- 40-Hour Week
  - Sustainable Pace
Whole Team (On-Site Customer)

- XP relies on a project community
- The project goals are a shared responsibility
- Development is an ongoing conversation across the whole team
- “A cooperative game of invention and communication” – Cockburn
- Face-to-face communication is the most efficient
- Team members available to others with minimal effort, waiting, and disruption
- Colocation facilitates whole team
Planning Game (Release & Iteration Planning)

- Facilitates incremental project planning as more and better information learned
- Develop rough plan first, refine incrementally
- XP designed to more quickly converge on an accurate plan based on real performance
- Release planning sets longer-term goals
  - Releases are typically from 1 to 6 months
- Iteration planning sets short-term time-box
  - Iterations are typically from 1 week to 1 month
Small Releases

- Releases small as possible while still delivering enough value to be worthwhile
- Release early to begin providing business value early (maximize ROI over time)
- Release early to obtain feedback and facilitate steering
- Small releases minimize early commitment, leaving open options longer
Acceptance Testing

- Customer writes acceptance tests
- Acceptance tests prove the system implements the desired features correctly
- Ideally acceptance tests written along with stories and provided prior to implementation
- Strong preference to automated tests
- All acceptance tests executed in each iteration (regression testing)
- Acceptance tests provide non-ambiguous specifications of functional requirements
- % of acceptance tests passing measure release completion
- Contributes to avoidance of quality debt
Unit Testing

- Developer writes unit tests
- Tests wrap each method that could break, with sufficient coverage
- Unit tests must be automated, clear pass/fail
- All unit tests executed very frequently
- Code can not be checked-in until all unit tests pass
- Unit tests provide safety net for refactoring
- Unit tests document intent and usage of code modules and methods
- Most languages have an xUnit framework (e.g., jUnit)
- Contributes to avoidance of quality debt
Test-Driven Development

- Likely the most innovative XP practice
- Developer writes a unit test prior to writing code
- Developer writes just enough code to make the unit test pass
- Developer refactors to ensure code is simple and clean (standards met)
- Developer continues until acceptance test passes

Test-driven-development tends to:
- Result in a remarkably complete set of tests
- Drive the code to be the most simple, minimal
- Leave clear examples of intent and use
Simple Design

- Design is an on-going activity
- Design in XP is not static – is incremental and responds to learning
- “Do the simplest thing that can possibly work” at each point:
  - Passes all known tests
  - Contains no duplication
  - Communicates intent clearly
  - Is a small as possible – fewest classes, methods, lines of code
- Does *not* mean “Do the stupidest thing…”
- No speculative development (YAGNI)
- Simple designs reduce friction and surface area
Pair Programming

- All production code written in pairs
- Programming as a collaborative conversation
- Pairing is not one person looking over the other’s shoulder
- Pairing provides (partial list):
  - All design decisions made by at least two
  - Continuous code reviews
  - Cross-training and mentoring spreads with pairs
  - Eases specialization dependencies & bottlenecks
  - Motivation as well as sanity checks, builds teamwork
- Research shows pairing can be more effective than solo programming
- Short-term costs made up by long-term benefits
- Contributes to avoidance of knowledge debt
Refactoring (Design Improvement)

- Improve the design of existing code without changing its functionality
- Allows design to incrementally evolve
- Refactoring not random, driven by learning from new implementations
- Refactoring can occur just prior or just after writing new code
- Refactoring drives code towards higher-level design patterns
- **Not** a substitute for hacking first, thinking later
- Contributes to avoidance of design debt (aka “cruft”)
Continuous Integration

- Avoidance of “big bang” integrations
- Integration for each pair occurs several times each day
- All tests run prior to commitment to code base
- Makes the cause of failures more obvious
- Minimizes merge pain
- Facilitates the code base evolving steadily
- Forces bug fixing to occur immediately
- Often supplemented by daily builds
- Contributes to the avoidance of quality & integration debt
Collective Ownership

- Any Developer can make changes to any part of the code as needed for their tasks
- Eliminates queuing bottlenecks
- All Developers responsible for integrity of the code base
- “You break it, you fix it” encourages collective responsibility
Coding Standards

- Consensus of coding style and practices
- Facilitates moving about the code base
- Contributes to definition of clean code and “doneness”
- Removes distraction of endless arguments
- Goal is that code looks anonymous
- Standards evolve over time
- Often expanded to non-coding aspects
For the team to communicate, it needs:
- A common mental model of the system
- A common language to talk about the system

The ideal model and language is that of the actual problem domain (Domain-Driven Design)

The ideal model evolves, and can take time

A full model has multiple levels and facets

A metaphor can provide a short-cut to reach the common model and language goals

“The payroll system is like an assembly line, where each deduction is bolted on to the paycheck as it moves down the line.”
Sustainable Pace (40-Hour Week)

- Development is a marathon, not a sprint
- Fatigue and stress reduces productivity
- Consideration of the human (humane) side
- Manager accountable for setting reasonable environment and project expectations
- Team agrees on expectations and enforces
- Exact hours not as important as ability to perform
- Contributes to avoidance of decompression debt
Additional Practices

- Stand-Up Meetings
- Tracking & Metrics
- Retrospectives
- Big Visible Charts
- Team Culture
- Consensus
- Skunk Works, War Room
- Version & Configuration Management, Automated Builds, Build Promotion
User Stories

- A story is a token for a piece of system capability to be implemented (sort of a change request)
- A story is a placeholder for a long-running conversation about requirements

Stories vs. Use Cases

- A story is not a use case, a use case is not a story
- A use case defines functional requirements in total
- Set of use cases define breadth and depth of system behavior, augmented with non-functional requirements
- A story points to a portion of breadth and depth
- Stories incrementally build up full scope
Project Management

- Release and Iteration Plans
  - Total stories capture overall scope
  - Plans capture sequencing and delivery milestones
- Story Estimating
  - Story estimates are approximations, expressed in “points”
  - Estimates eventually fall in probability ranges
  - Often are expressed in ideal-time units
- Task Estimating
  - Optionally used to check story estimates
- Tracking
  - Basic tracking is velocity – story points per iteration
  - Optional actual vs. estimate tracking
  - Optional points per developer time correlation
Tracking Example

“Burn-Up” Chart (place in visible location)
Agenda

- Agility
- History of Extreme Programming
- XP and the World of Agile Processes
- XP Overview
- XP Project Community
- XP Process Cycle
- XP Practices
- Other XP Topics
Scaling XP

- XP seems to allow smaller teams to accomplish an awful lot.
- XP seems to hit single-team challenges around 12-16 developers.
- XP can scale by building recursive teams.
- Recommended to build small team first, incrementally grow, and use first team to seed recursive teams.
- XP has been used on teams of 40-50.
Documentation

- XP is a minimalist process
- Many teams successful with index cards and large Post-Its®
- XP is not anti-documentation, but encourages doing the least amount that is really needed
- Document when needed for distributed sharing, historical needs, summarizing, etc.
Adopting XP

- Find a suitable team and project
- Two strategies:
  - Adopt XP by-the-book first, then modify
  - Refactor your process incrementally to XP
- Be mindful of practice synergies
- Manage the team culture and change
- Allow sufficient time for feedback and learning
- An experienced coach can be valuable
Where Does XP Go From Here?

- Definition of Customer practices
- Integration with other disciplines
  - Database
  - Interface Architecture/Usability
- Relationship of roles to existing job functions
- Theoretical grounding of practice synergy
- XP has successfully climbed the early adopter slope
- XP has reached the edge of the “chasm”
Resources

- Lots of books – come see them

- Web sites:
  - http://www.xprogramming.com/
  - http://www.extremeprogramming.org/
  - http://www.agilelogic.com/resources.html

- Discussion Forums:
  - http://groups.yahoo.com/group/extremeprogramming/
  - http://groups.yahoo.com/group/xpsocal/