Radically better software development with Extreme Programming

Carl Erickson
Atomic Object LLC
October 2002
The software crisis

• Software is all too often
  – Over budget
  – Late to market
  – Buggy
  – Not accepted by users
  – Not extensible in maintenance

• The unbelievable statistic

• Nato conference, 1968
Successful software projects

- Solves real business problem
- Satisfies users
- Acceptable cost per budget and business case
- Met all deadlines
- Produced software that is
  - stable, not buggy,
  - has acceptable performance
  - extensible, maintainable, remains malleable as it ages
Technology to the rescue!

- high level languages
- structured development
- 4GL
- AI
- client/server
- OO
- visual programming
- Java
- .NET
- AOP
Human activity

- Software is produced by teams of people
- Technologies alone are an unlikely savior
- It's not the tools we use, it's the way we use them and how we interact
Heavyweight methodologies

• Logical reaction to the sad state of affairs
• Goal: define a rigorous, quantifiable development process, follow it
• Examples include
  – SEI Capability Maturity Model
  – Rational Unified Process,
  – ISO 9001
Heavyweight characteristics

• Emphasis on artifacts (diagrams, models, documents) and formal communication
• Gives managers something concrete to do, control, and believe in
• Heavyweight, proscriptive, anti-creative, high overhead, hated by developers
The unquestioned Truth

Up-front requirements analysis, design, and modeling are the best ways to avoid disaster.
Origins and problems

• The high cost of change late in a project
• Underlies most of the heavyweights
• The trouble is, this unquestioned truth is a based on a flawed assumption

• For all but trivial applications, the actual requirements, even the real problem, are unknowable in advance, out of context
Users and monkeys

• Users can't tell what they want, or how it'll work, until they have something to try, in context

• Monkey behavior

• Editing vs Writing

• Satisfying requirements vs Solving real problems
Tragic mismatch

• Heavyweighters are:
  rational,
  require prior knowledge,
  scientific,
  linear,
  isolated

• Software development is:
  chaotic,
  unpredictable,
  craft-like,
  cyclical,
  contextual
Lightweight processes

- Rules, process, forms, statistics - do not equate to quality
- Developer driven reaction to heavyweight
- Start from known best practices (bottom up)
- Respect and consider the craft, i.e. the skill and human aspects, of software development
- Every situation is different, so the process must flex, be agile
- XP is one of several (Scrum, UCD, RAD, Crystal, etc)
Extreme Programming

2am, marathon, Cheetos hacking, greasy, undocumented, brilliant what tests?, mysterious Mountain Dew, protective ad-hoc, brittle, cubicle fear change, cowboy overly elegant, code ego real soon now, difficult to talk to
Extreme Programming

2am, marathon, Cheetos hacking, greasy, undocumented, brilliant, what tests?, mysterious, Mountain Dew, protective ad-hoc, brittle, cubicle fear change, cowboy overly elegant, code ego real soon now, difficult to talk to

8-5, teams and pairs, code reviews, communication, coding standards, snacks, communal code tests, tests, tests, maintainable predictable copes with change
A rose by any other name...

- Names are important (objects, classes, methods, variables, titles, roles, companies)
- Atomic Object
- XP: silly name, sound basis
Brief history

- Kent Beck, Ron Jeffries, Ward Cunningham, Martin Fowler
- Late 1990s
- Michigan connection - Chrysler payroll
- Conferences
- Books, references
  - http://atomicobject.com/
The twelve practices

- Planning
- Simple design
- Testing
- Pair programming
- Collective ownership
- Continuous integration

- Refactoring
- On-site customer
- Coding standards
- 40 hour week
- Small releases
- Metaphor
Testing

• Terminology: unit, integration, system
• Test first design
  – just-in-time specification
  – catalyst for a two-way design conference and opportunity for expert consultation
  – the first (and ongoing) client of the software
  – as a measure of when you're done coding
  – documentation of source
Tests and code bulk
Regression testing

• Unit and integration regression testing
  – needs to be automated
• Lightweight framework to support this
  – Composite pattern, parallel test hierarchy
    – JUnit, CppUnit, etc
• Design for test often leads to better design
• The gift that keeps on giving
Design for test example
Easier test, better design
Test infected

- A good disease to catch; contagious too
- Small immediate, positive feedback from having tests pass
- Confidence in releasing code that works (ego and pride)
Acceptance testing

• Tests from the customer/user perspective
• Written by the customer
• How the customer knows you did what you committed to do
• Automation
  – Atomic Object Haste framework
Pair programming

- Two people, one computer
- Roles: driver, navigator
- Pair programming as extreme code review
- Spreads knowledge and expertise
  - avoid single point of expertise
  - great for training/mentoring
- Formation
Pair programming studies

• Laurie Williams, NC State University
  – Pairs solve problems faster (wall clock time)
  – Code produced is simpler, better
  – Overhead (person hours) is around 20%

• My observations
  – Getting stuck less often
  – Becoming distracted (SlashDot effect)
  – Challenged by another POV
  – Following good practices
Facilities

• Single computer per pair (cost savings!)
• Wide tables without legs
• Open environment (vs pairs in cubicles)
On-site customer

- Communication with developers:
  - high bandwidth,
  - low latency,
  - direct communication

- Customer specifies, or even writes, acceptance tests

- How it’s worked for Atomic Object
Synergy

• Picking XP apart, one practice at a time
  – “we do that already”
  – “that [alone] won’t solve the problem”
  – “we can’t do that”

• The whole greater than the sum of the parts
Testing & Refactoring & Simple Design

- Simple, as needed, design will need refactoring when the code starts to smell
- Refactoring is scary
- Test suites tell you what is expected
- Regression testing tells you if you make a mistake
AO Development Cycle
Testing & Pair Programming

• What you might not think of alone
• Keeping it simple
• What you might not feel like doing (until you’re infected)
• What you might not be good at
Collective Ownership & Coding Standards & Pair Programming

- Reading other peoples code
- Avoiding religious battle concerning {}
- Knowing about the whole code base
- Modifying what you didn't write
Radical impact

• Improving software quality
  – Happy users, good performance
  – Maintenance and extensibility

• Coping with change
  – Moving faster with confidence
  – Discovering new opportunities
  – Strategic advantage
Radical impact, continued

• Reducing risk
  – Growing a complex system from a simple system
  – Discovering and solving real problems

• Happy developers
  – Producing quality, tested software systems
  – Working together
  – Avoiding over specialization
  – No death marches and mandatory overtime
Values

- Communication
- Simplicity
- Feedback
- Courage