

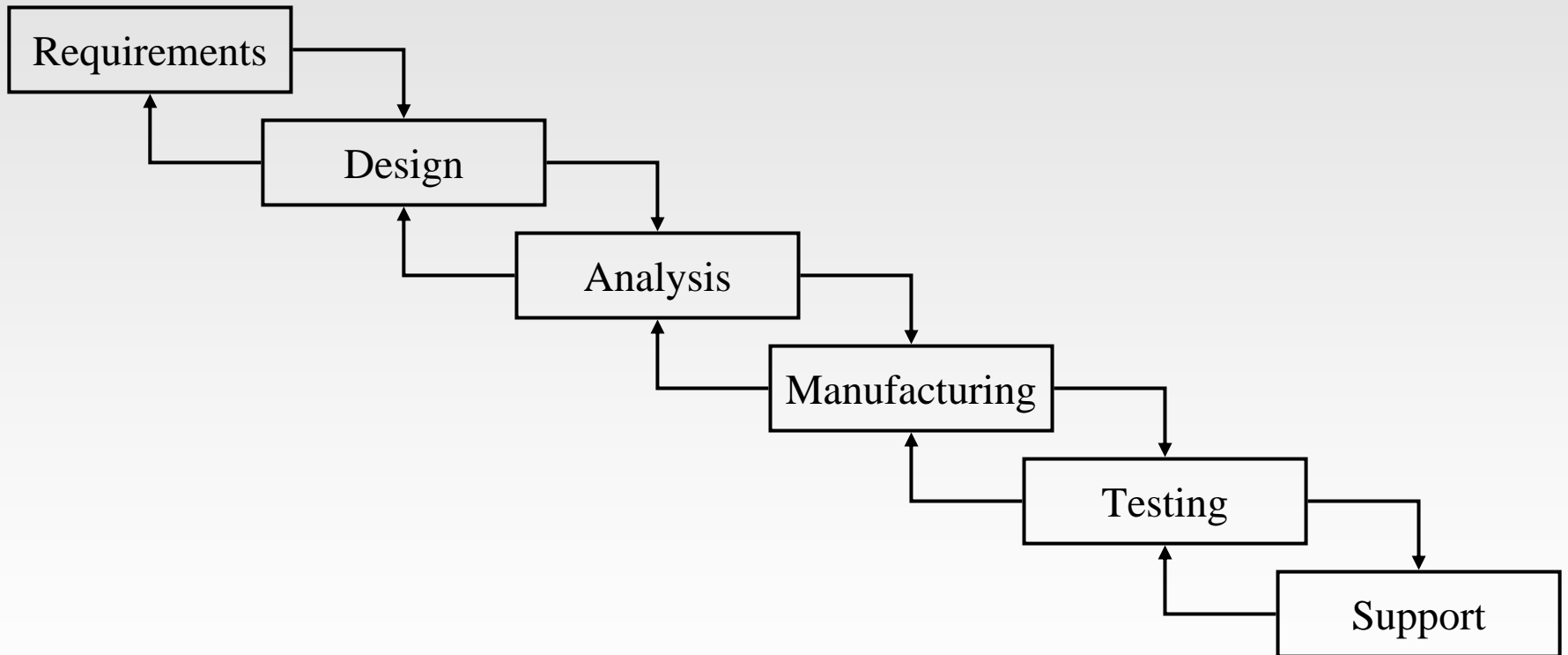
Software Development Methodology

Typical Design Process Models

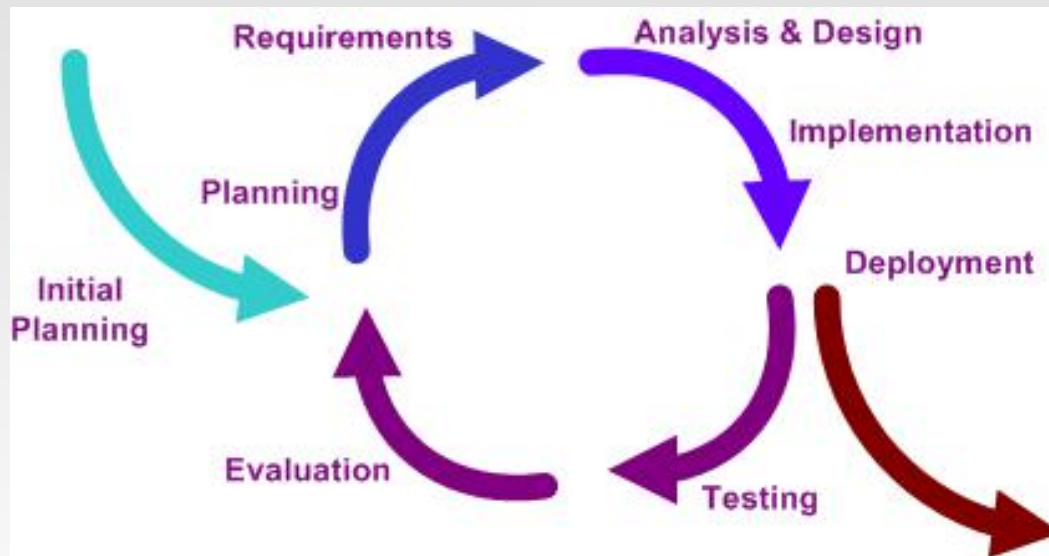
- Waterfall
- Iterative
- V Model
- Spiral
- Agile Methods



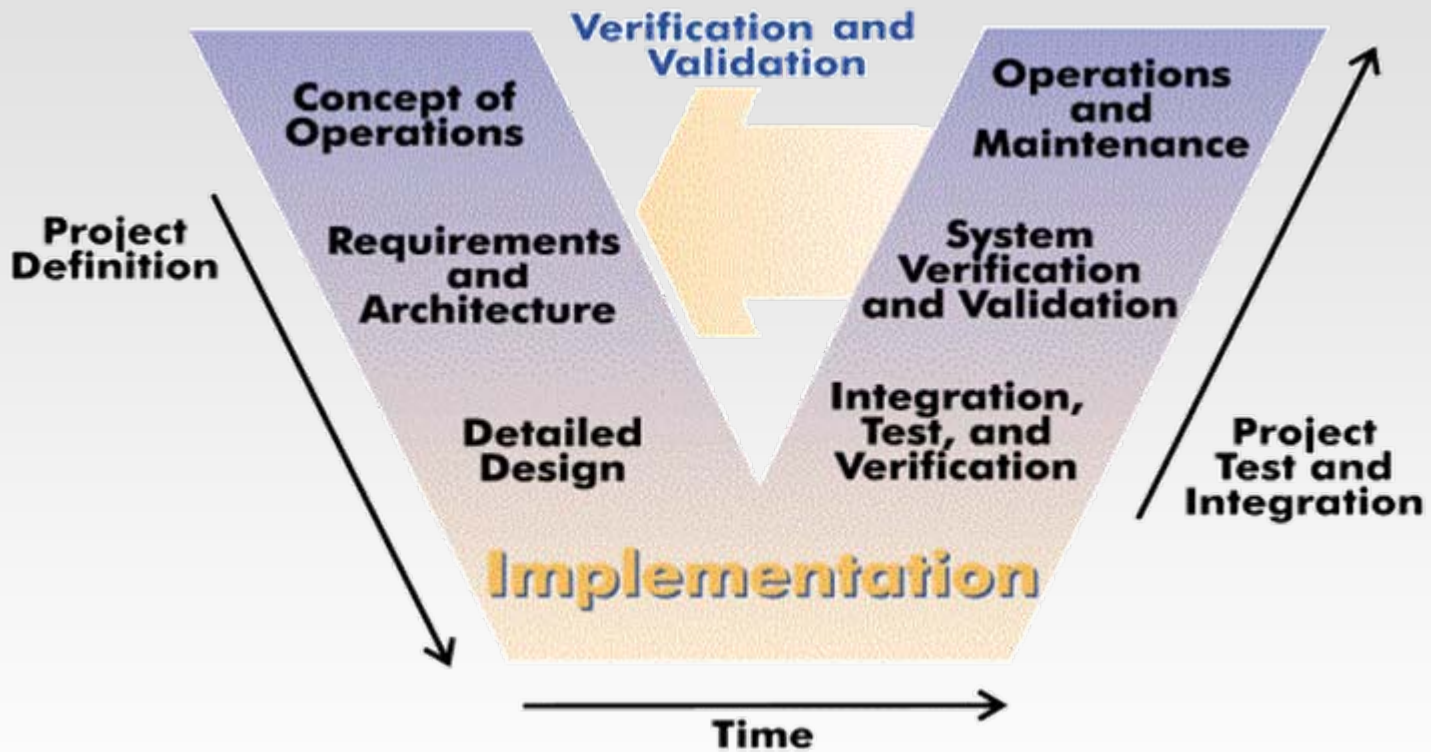
Waterfall



Iterative Development Model

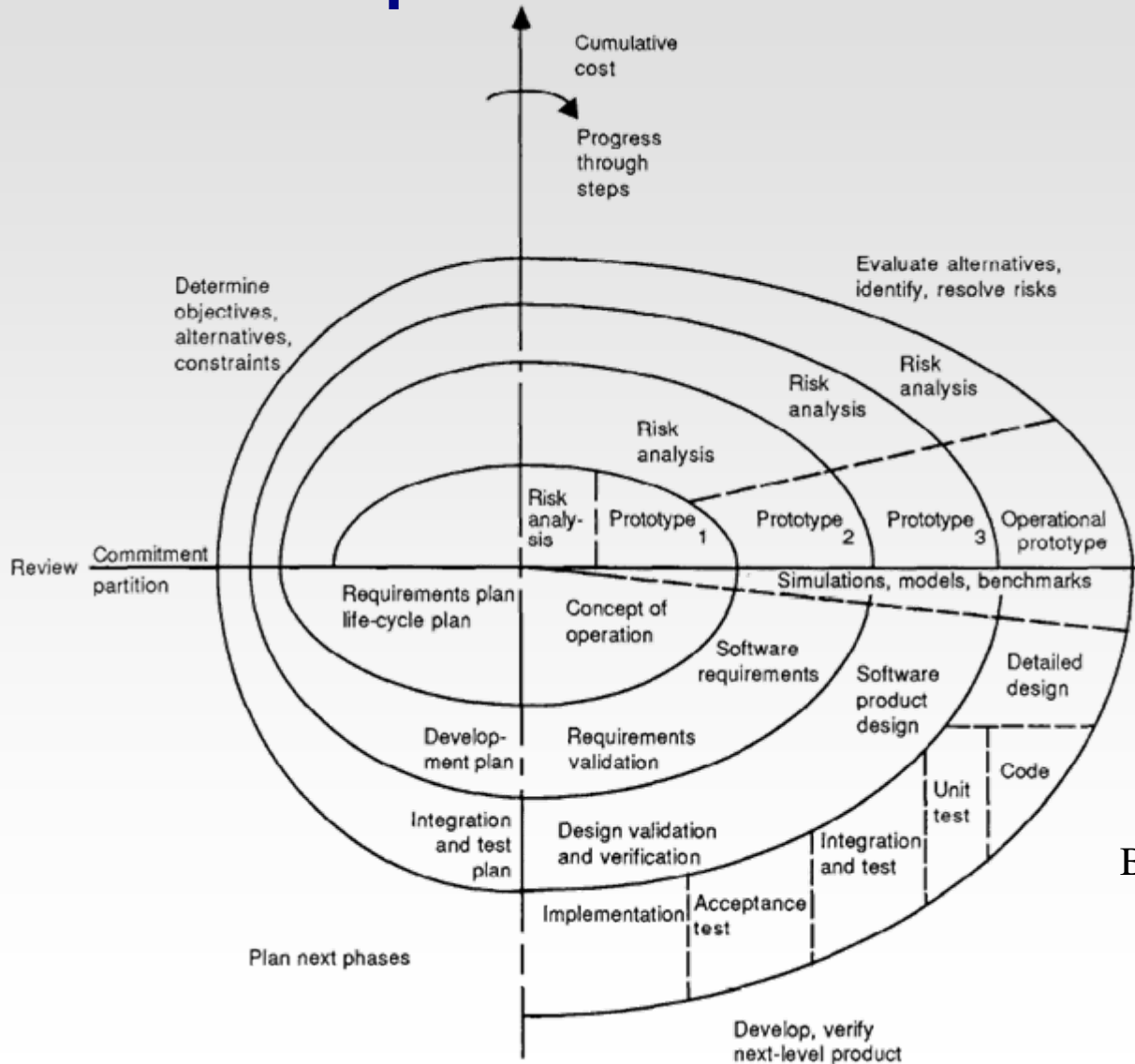


V Model



Osbourne, *et al*, 2005

Spiral Model



Boehm, 1988

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it.
Through this work we have come to value:

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 is value in the items on the right,
we value the items on the left more.

“The Agile Alliance” – Feb 2001

Extreme Programming (XP)

- Developed in mid '90s at Chrysler by Kent Beck and colleagues
- Tried to address issues such as requirements creep, schedule slippage, software bloat, quality, etc...

Core Values

- Communication
- Simplicity
- Feedback
- Courage

Core Activities

- Coding
- Testing
- Listening
- Designing



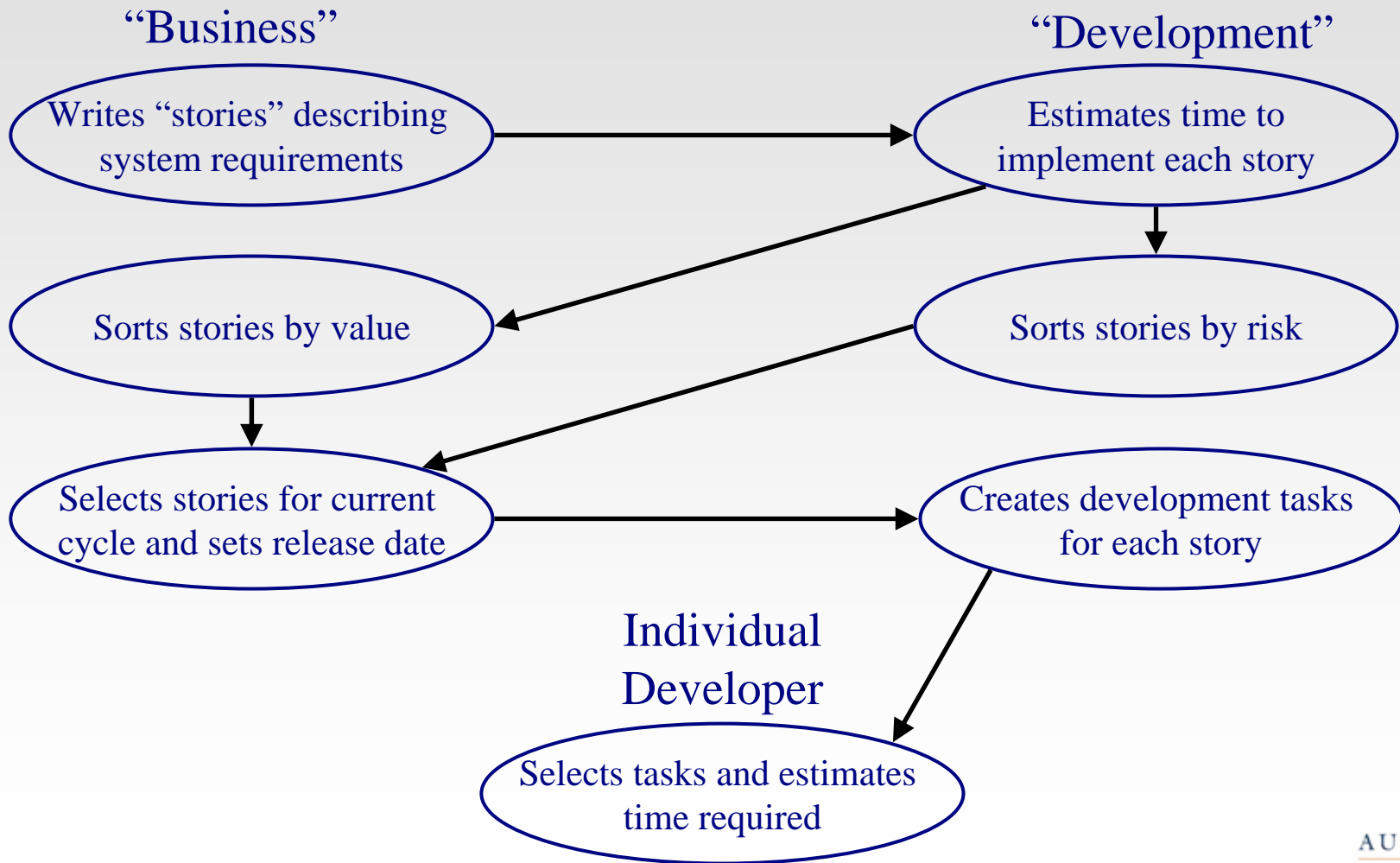
Putting the Extreme in XP

Effective Programming Elements

- Code review -> review all the time
- Testing -> test all the time
- Design -> redesign all the time
- Simplicity -> major focus/goal
- Architecture -> refine all the time
- Integration testing -> integrate and test continuously
- Short iterations -> hours not months



XP Planning Example



XP Development Example

- Developer selects task card and identifies appropriate team member to pair with for task
- Pair writes test cases to verify new functionality
- Pair writes code to implement new functionality
- Pair integrates new code into software base
- Pair runs through all test cases to regression test new code



Feature/Requirements Creep

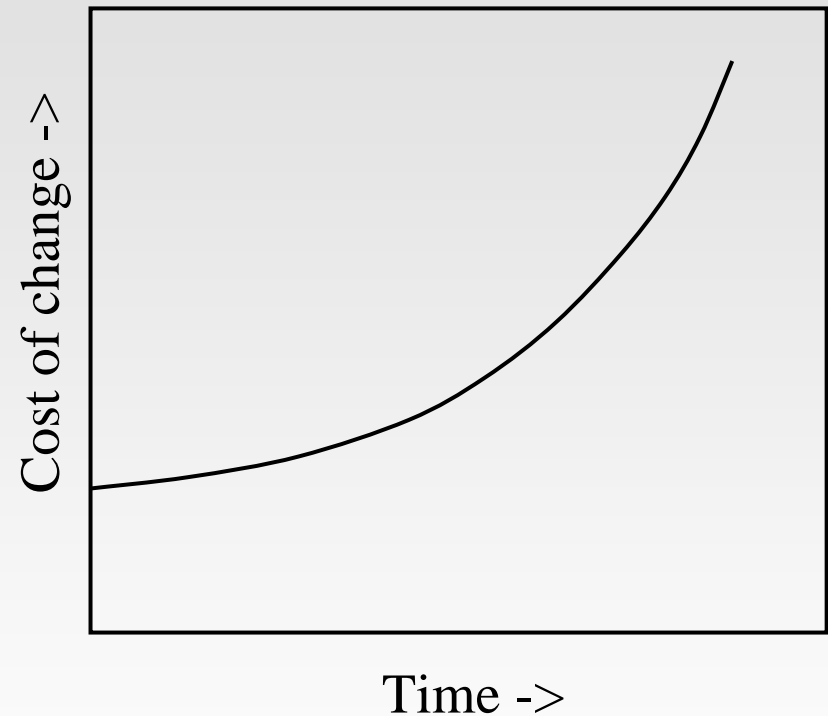
- Requirements always change over time and this is a major contributor to schedule slips
- Significant effort is expended designing for and implementing requirements that aren't needed
- Significant costs are incurred changing the design late in the process to implement new requirements
- What if changing the design, even late in the process didn't cost so much?



Cost of Design Changes

As time progresses, the cost of making a change in design increases dramatically

- More design is complete, more things will be affected by change
- Change may invalidate previous analyses/tests
- Configuration management
- Logistics tail



XP Reduces Cost of Design Changes

XP reduces cost of design changes at any stage of development

- Analysis/testing is automated so design can easily be retested for compliance with requirements
- Reduces risk of unintended consequences

If design changes don't cost that much, even late in the game, don't bother expending resources designing/developing features until they are really needed

YAGNI – You ain't gonna need it!

XP Practices

1. The Planning Game
2. Small Releases
3. Metaphor
4. Simple Design
5. Testing
6. Refactoring
7. Pair Programming
8. Collective Ownership
9. Continuous Integration
10. 40 hr Weeks
11. Customer Onsite
12. Coding Standards



XP Practices 1: The Planning Game

- A dialog between customer and development team
- Customer or customer representative is required to be on site with development team
- Resolves conflicts between business considerations and technical considerations

Business

- Scope
- Priority
- Composition of releases
- Dates of releases

Technical

- Estimates
- Consequences
- Process
- Schedule



XP Practices 2: **Small Releases**

Small releases are scheduled every 1-2 months

- Easier to plan 1-2 months ahead rather than longer periods
- Facilitated by testing approach



XP Practices 3: **Metaphor**

- A single description of the system that is meaningful to all participants
- A language or vocabulary for describing the components of the system and how the system works and how the components work together



XP Practices 4: Simple Design

Design is correct if it

1. Runs all of the tests
2. Contains no duplicate code
3. Expresses important intentions to programmers
4. Contains fewest possible classes and methods

“Simplicate and add more lightness”
– Bill Stout



XP Practices 5: Testing

- Test driven development paradigm
- Every feature of the system must have a test
- Programmers write test cases first before writing production code
- Anytime a bug is found, a test is written to duplicate it prior to fixing it
- Testing is automated and run frequently



Test Driven Development

- Used as early as 1950's by NASA's project Mercury
- Partially used in certification today – certain tests are dictated by FARs and manufacturers attempt to meet those requirements as cheaply as possible
- Functional tests are similar to constraints in MDO analysis



XP Practices 6: Refactoring

- Modifying code to make adding a feature easier or to simplify code, or to remove or avoid duplicate code
- Supported by testing approach so programmer can immediately see if something broke



XP Practices 7: Pair Programming

- Two people always work together on one machine when coding
- Form of design review
- Dynamic pairing, not always the same person
- Aids intra-team communication and enhances addition/distribution of corporate knowledge



XP Practices 8: **Collective Ownership**

- Anyone can change/add to any of the code
- Special requirements are expressed in test cases not buried in specs
- Everyone is responsible for the code base



XP Practices 9: Continuous Integration

- New code is integrated and tested within a few hours of development
- New code must pass all tests
- If code doesn't pass tests – person responsible is easily identified



XP Practices 10: 40 hour Weeks

- Fresh and eager every morning, tired and satisfied every night
- Sustainable pace, not sprint and collapse
- As a rule team can't work a second weekend of overtime



XP Practices 11: Onsite Customer

- Resolves questions about business needs
- Receives feedback from team on technical issues and ramifications of requirements



XP Practices 12: Coding Standards

- Team must agree on coding standards
- Collective code ownership requires agreement on coding standards to avoid inefficiencies of constant battles over code formatting and coding styles



Application to Your Project

- #1 – Planning Game
 - “Stories” ~ Use Cases
 - Structuring use cases with goals
 - Use cases, ten years later
- #2 – Small Releases
- #4 – Simple Design
- #5 – Testing
- #8 – Collective Ownership
- #9 – Continuous Integration
- #11 – Customer Onsite
- #12 – Coding Standards

