Software Engineering
Rochester Institute of Technology

Introduction to eXtreme Programming (XP)

Rochester Institute of Technology
Software Engineering Department

Extreme Programming (XP)

- ✓ Kent Beck – "C3 Project" – Chrysler Comprehensive Compensation system.

- ✓ XP Values:
  - *Communication*
  - *Courage*
  - *Feedback*
  - *Simplicity*
  - *Respect (2nd edition)*

- ✓ Established the Twelve Practices

Four Project Variables

- ✓Time – duration of the project
- ✓Quality – the requirements for ‘correctness’
- ✓Resources – personnel, equipment, etc.
- ✓Scope – what is to be done; the features to be implemented

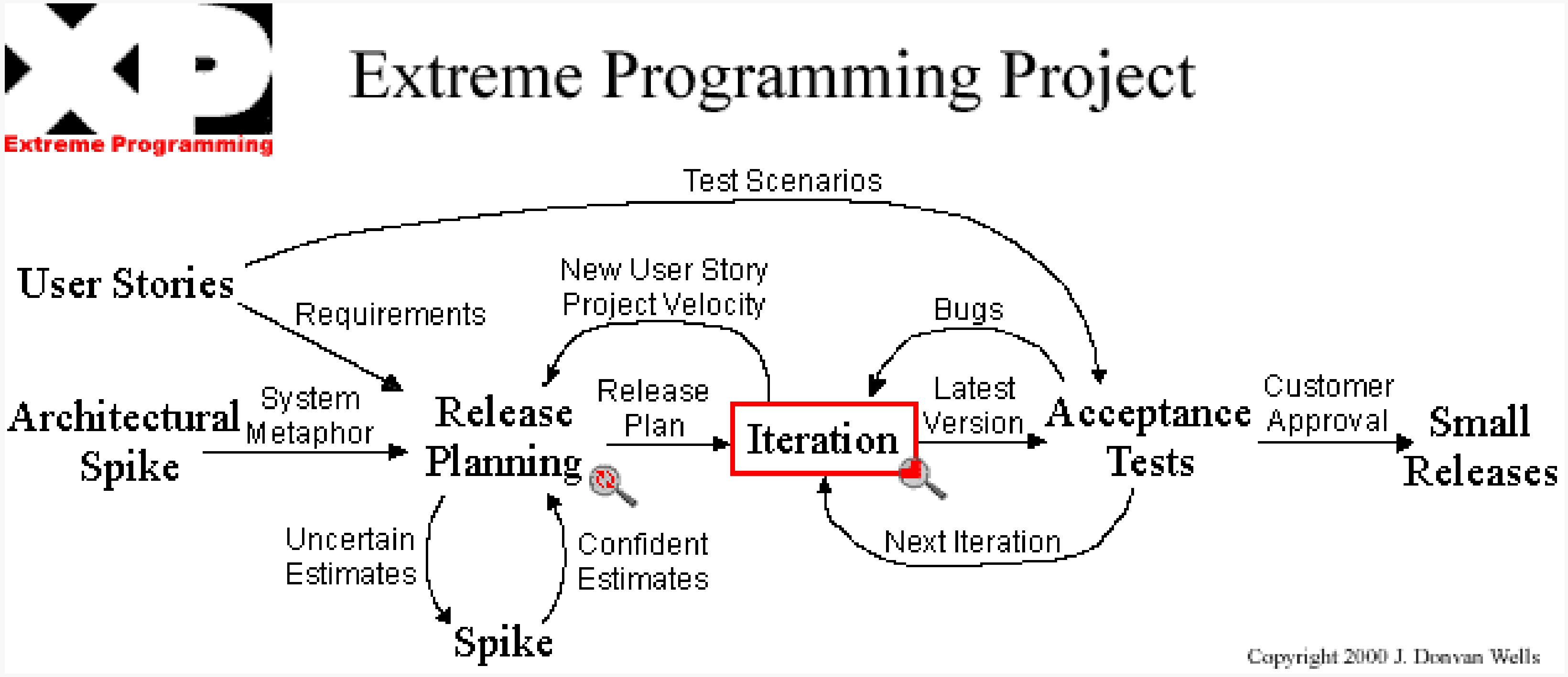
- ✓Pick three, any three . . .

Software Engineering
Rochester Institute of Technology

Original Twelve Practices (XP)

- ✓ Metaphor
- ✓ Release Planning
- ✓ Testing
- ✓ Pair Programming
- ✓ Refactoring
- ✓ Simple Design
- ✓ Collective Code Ownership
- ✓ Continuous Integration
- ✓ On-site Customer
- ✓ Small Releases
- ✓ 40-Hour Work Week
- ✓ Coding Standards

The Extreme Lifecycle



from “Extreme Programming: a gentle introduction”

http://www.extremeprogramming.org/

Software Engineering
Rochester Institute of Technology

The 12 Practices of XP

1. Metaphor
2. Release Planning
3. Testing
4. Pair Programming
5. Refactoring
6. Simple Design
7. Collective Code Ownership
8. Continuous Integration
9. On-site Customer
10. Small Releases
11. 40-Hour Work Week
12. Coding Standards

Software Engineering
Rochester Institute of Technology

Metaphor

- ✓The closest XP comes to architecture
- ✓Gives the team a consistent picture of describing the system, where new parts fit, etc.
- ✓C3 payroll . . . The paycheck goes down the assembly line and pieces of information are added.
- ✓Sometimes, you just can’t come up with one

Software Engineering
Rochester Institute of Technology

Release Planning

- ✓ Requirements via User Stories
  - ➢ *Short (index-card length) natural language description of what a customer wants (A commitment for further conversation)*
  - ➢ *Prioritized by customer*
  - ➢ *Resource and risk estimated by developers*
- ✓ Via “The Planning Game”
  - ➢ *Highest priority, highest risk user stories included in early “time boxed” increments*
- ✓ Play the Planning Game after each increment

Testing

- ✓ Test-Driven Development (TDD)
  - ➢ *Write tests before code*
  - ➢ *Tests are automated*
  - ➢ *Often use xUnit framework*
  - ➢ *Must run at 100% before proceeding*
  - ➢ *Great example of XP style TDD with Bob Martin:* *http://www.objectmentor.com/resources/articles/xpepisode.htm*
- ✓ Acceptance Tests
  - ➢ *Written with the customer*
  - ➢ *Acts as "contract"*
  - ➢ *Measure of progress*

Software Engineering
Rochester Institute of Technology

Pair Programming

Pair-programming has been popularized by the eXtreme Programming (XP) methodology

With pair-programming:

•Two software engineers work on one task at one computer

•One engineer, the driver, has control of the keyboard and mouse and creates the implementation

•The other engineer, the navigator, watches the driver's implementation to identify defects and participates in on-demand brainstorming

•The roles of driver and observer are periodically rotated between the two software engineers

Software Engineering
Rochester Institute of Technology

Research Findings to Date

✓Strong anecdotal evidence from industry

- *"We can produce near defect-free code in less than half the time."*

✓Empirical Study

- *Pairs produced higher quality code*
  - 15% more test cases passed (difference statistically significant)
- *Pairs completed their tasks in about half the time*
  - 58% of elapsed time (difference not statistically significant)
- *Most programmers reluctantly embark on pair programming*
  - Pairs enjoy their work more (92%)
  - Pairs feel more confident in their work products (96%)

Refactor Mercilessly

- ✓ **Improve the design of existing code without changing functionality**
  - ➢ ***Simplify code***
  - ➢ ***Opportunity for abstraction***
  - ➢ ***Remove duplicate code***

- ✓ **Relies on testing to ensure nothing breaks in the process of refactoring.**

Simple Design

- ✓ No Big Design Up Front (BDUF)
- ✓ "Do The Simplest Thing That Could Possibly Work"
  - ➢ *Including documentation*
- "You Aren't Gonna Need It" (YAGNI)
- ✓ CRC cards (optional)

- ✓ Technical Debt
  - *Total amount of less-than-perfect design and implementation decisions in your project*
  - *XP takes a fanatical approach to reducing technical debt via simple design and refactoring*

Collective Code Ownership

- ✓Code to belongs to the project, not to an individual engineer
- ✓As engineers develop required functionality, they may browse into *and modify* any class.

Continuous Integration

- ✓ Pair writes up unit test cases and code for a task (part of a user story)
- ✓ Pair unit tests code to 100%
- ✓ Pair integrates
- ✓ Pair runs ALL unit test cases to 100%
- ✓ Pair moves on to next task with clean slate and clear mind
- ✓ Should happen once or twice a day.
- ✓ Prevents IntegrationHell

On-Site Customer

- ✓ Customer available on site to clarify stories and to make critical business decisions.
  - *Product managers, domain experts, interaction designers, business analysts*
  - *Ideally 2 “customers” for every three programmers*
- ✓ Developers don’t make assumptions
- ✓ Developers don’t have to wait for decisions
- ✓ Face to face communication minimizes the chances of misunderstanding

Software Engineering
Rochester Institute of Technology

Small Releases

- ✓Timeboxed
- ✓As small as possible, but still delivering <u>business value</u>
  - ➢*No releases to 'implement the database'*
- ✓Get customer feedback early and often
- ✓Do the planning game after each iteration
  - ➢*Do they want something different?*
  - ➢*Have their priorities changed?*

Software Engineering
Rochester Institute of Technology

Sustainable Pace

- ✓Kent Beck says, “ . . . fresh and eager every morning, and tired and satisfied every night”
- ✓Burning the midnight oil kills performance
- ✓Tired developers make more mistakes, which slows you down more in the long run
- ✓If you mess with people’s personal lives (by taking it over), in the long run the project will pay the consequences

Coding

- ✓Use Coding Conventions
  - ➢*Considering Pair Programming, Refactor Mercilessly, and Collective Code Ownership . . . need to easily find your way around (other people's) code*
- ✓**Method Commenting**
  - ➢*Priority placed on* intention-revealing code
    - ➢**If your code needs a comment to explain it, rewrite it.**
    - ➢**If you can't explain your code with a comment, rewrite it.**

The 13th Practice? The Stand Up Meeting

✓Start day with 15-minute meeting
- ➢*Everyone stands up (so the meeting stays short) in circle*
- ➢*Going around the room everyone says specifically:*
  - ➢What they did the day before
  - ➢What they plan to do today
  - ➢Any obstacles they are experiencing
- ➢*Can be the way pairs are formed*

2nd Edition of XP

- ✓ **Practices divided into "primary" and corollary"**
- ✓ **Primary practices**
  - ***Sit together***
  - ***Whole team***
  - ***Information workspace***
  - ***Energized work***
  - ***Pair Programming***
  - ***Stories***
  - ***Weekly cycle***
  - ***Quarterly cycle***
  - ***Slack***
  - ***10 minute build***
  - ***Continuous integration***
  - ***Test First Programming***
  - ***Incremental design***

2nd Edition of XP

✓Corollary Practices

- *Real customer involvement*
- *Incremental deployment*
- *Team continuity*
- *Shrinking teams (frees people to form more teams)*
- *Root-cause analysis (see next slide)*
- *Shared code*
- *Code and test (only permanent artifacts)*
- *Single code base*
- *Daily deployment*
- *Negotiated scope contracts (time, cost, quality fixed)*

Root Cause Analysis

- ✓Every time a defect is found, eliminate the defect *and* its cause.

- ✓XP response to a defect:
  - *Write an automated system-level test that demonstrates the defect.*
  - *Write a unit test that also reproduces the defect*
  - *Fix the system so that the unit test passes (should also cause system test to pass)*
  - *Once the defect is resolved, figure out why the defect was created and wasn't caught.*

Software Engineering
Rochester Institute of Technology

XP/Scrum Cross Reference

✓ Collaboration
- *Sit together (XP) -> Open work environment (Scrum)*
- *Whole Team -> Scrum Team*
- *Stand-Up Meetings -> Daily Scrum*
- *Iteration Demo -> Sprint Review*

✓ Planning
- *Release Planning -> Product Backlog*
- *Iteration Planning -> Sprints*
- *Stories -> Backlog Items*

✓ In general…
- *XP leans towards development practices*
- *Scrum leans towards project management practices*

Resources

- *Agile Software Development Portal:*
  agile.csc.ncsu.edu/
- *Agile Alliance –* ***www.agilealliance.com***
- ***www.extremeprogramming.org/***
- *Laurie Williams – North Carolina State:* ***collaboration.csc.ncsu.edu/laurie/index.html***
- *Extreme Programming Explained – 2nd Edition, Kent Beck*
- *Agile Development, James Shore & Shane Warden*