
Go
free
RANGE

Pair Programming

Disclaimer

- Not a prescription
- Not a criticism
- Just our opinions

Overview

- What is pair programming?
- Why we think pair programming is a Good Thing™
- Why we sometimes find it hard to **start** pairing
- Why we sometimes find the **process** of pairing difficult
- Q&A

What is Pair Programming?

Two people attacking a single problem through code and discussion.

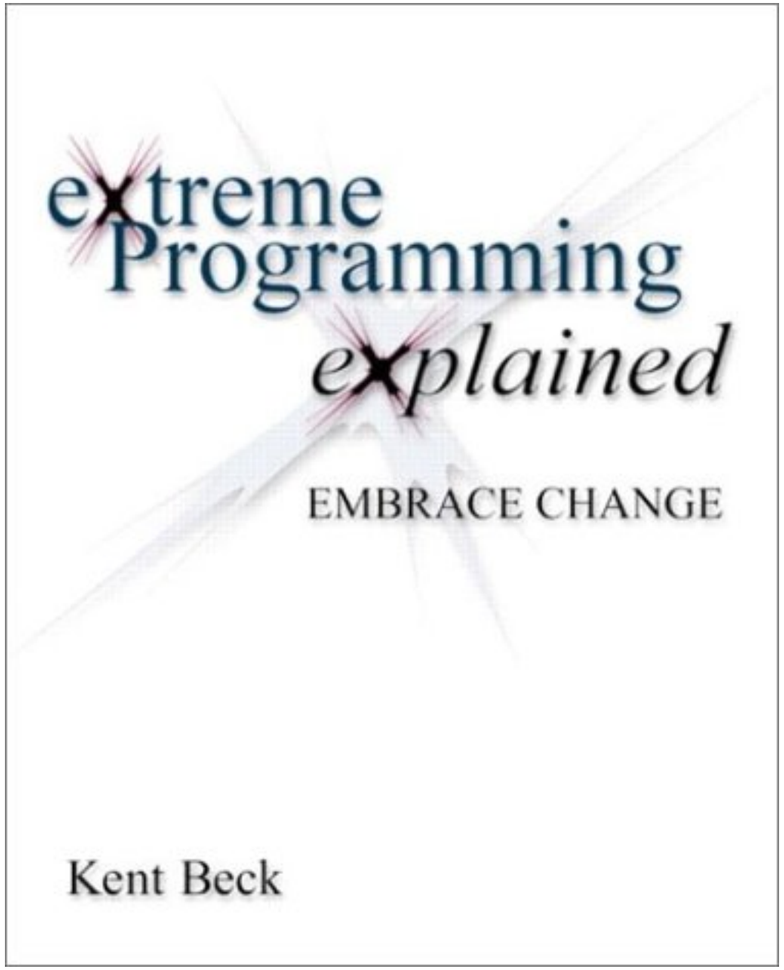
— Chris & James

What Pair Programming is Not

- Someone judging you on what you're doing.
- Something only less experienced people do until they're up to speed.
- Two people doing the job of one person.

What is Pair Programming?

XP: The "White Book"



All production code is written with two people looking at one machine, with one keyboard and one mouse.

There are two roles in each pair.

One partner, the one with the keyboard and the mouse, is thinking about the best way to implement this method *right here*.

The other partner is thinking *more strategically*.

Pairing is *dynamic*. If two people pair in the morning, in the afternoon they might easily be paired with other folks.

If you have responsibility for a task in an area that is *unfamiliar* to you, you might ask someone with *recent experience* to pair with you.

More often, anyone on the team will do as a partner.

What Pair Programming is Not

- One person programming while another watches.
- A one-way tutoring session.
- About being joined at the hip.

What is Pair Programming?

- A dialog between two people trying to simultaneously program.
- A conversation at many levels assisted by & focused on a computer.
- A subtle skill - you can spend the rest of your life getting good at.

Why we think pairing is a Good Thing™

XP Practices (1-6)

- The Planning Game
- Small Releases
- Metaphor
- Simple Design
- Testing
- Refactoring

XP Practices (7-12)

- Pair Programming
- Collective Ownership
- Continuous Integration
- 40-hour Week
- On-site Customer
- Coding Standards

XP Practices Supporting Pairing

You can't possibly write all the production code in pairs. It will be too slow. What if two people don't get along? Unless:

- The **coding standards** reduce the picayune squabbles.
- Everyone is **fresh and rested**, reducing further the chance of … uh … discussions.
- The pairs **write tests** together, giving them a chance to align their understanding before tackling the meat of the implementation.
- The pairs have the **metaphor** to ground their decisions about naming and basic design.
- The pairs are working within a **simple design**, so they can both understand what is going on.

Then perhaps you *could* write all production code in pairs.

Besides, if people program solo they are more likely to **make mistakes**, more likely to **overdesign**, and more likely **drop the other practices**, particularly under pressure.

XP Practices Supported By Pairing

Simple Design

You couldn't possibly have just enough design for today's code. You would design yourself into a corner and then you'd be stuck, unable to continue evolving the system. Unless:

- You were programming with a partner, **so you were confident you were making a simple design, not a stupid design.**

Then perhaps you could get away with doing the best possible job of making a design for today.

Testing

You couldn't possibly write all those tests. It would take too much time. Programmers won't write tests. Unless:

- You are programming with a partner, **so if you can't think of another test your partner can, and if your partner feels like not bothering with the tests, you can gently rip the keyboard away.**

Then perhaps programmers and customers will write tests. Besides if you don't write automated tests, the rest of XP doesn't work nearly as well.

Refactoring

You couldn't possibly refactor the design of the system all the time. It would take too long, it would be too hard to control, and it would most likely break the system. Unless:

- You program in pairs, **so you are more likely to have the courage to tackle a tough refactoring, and you are less likely to break something.**

Then perhaps you could refactor whenever you saw the chance to make the system simpler, or reduce duplication, or communicate more clearly.

Collective Ownership

You couldn't possibly have everybody potentially changing anything anywhere. Folks would be breaking stuff left and right, and the cost of integration would go up dramatically. Unless:

- You pair program, **so you are less likely to break code, and programmers learn faster what they can profitably change.**

Then perhaps you could have anyone change code anywhere in the system when they see the chance to improve it. Besides, without collective ownership the rate of evolution of the design slows dramatically.

Continuous Integration

You couldn't possibly integrate after only a few hours of work. Integration takes far too long and there are too many conflicts and chances to accidentally break something. Unless:

- You program in pairs, **so there are half as many streams of changes to integrate.**

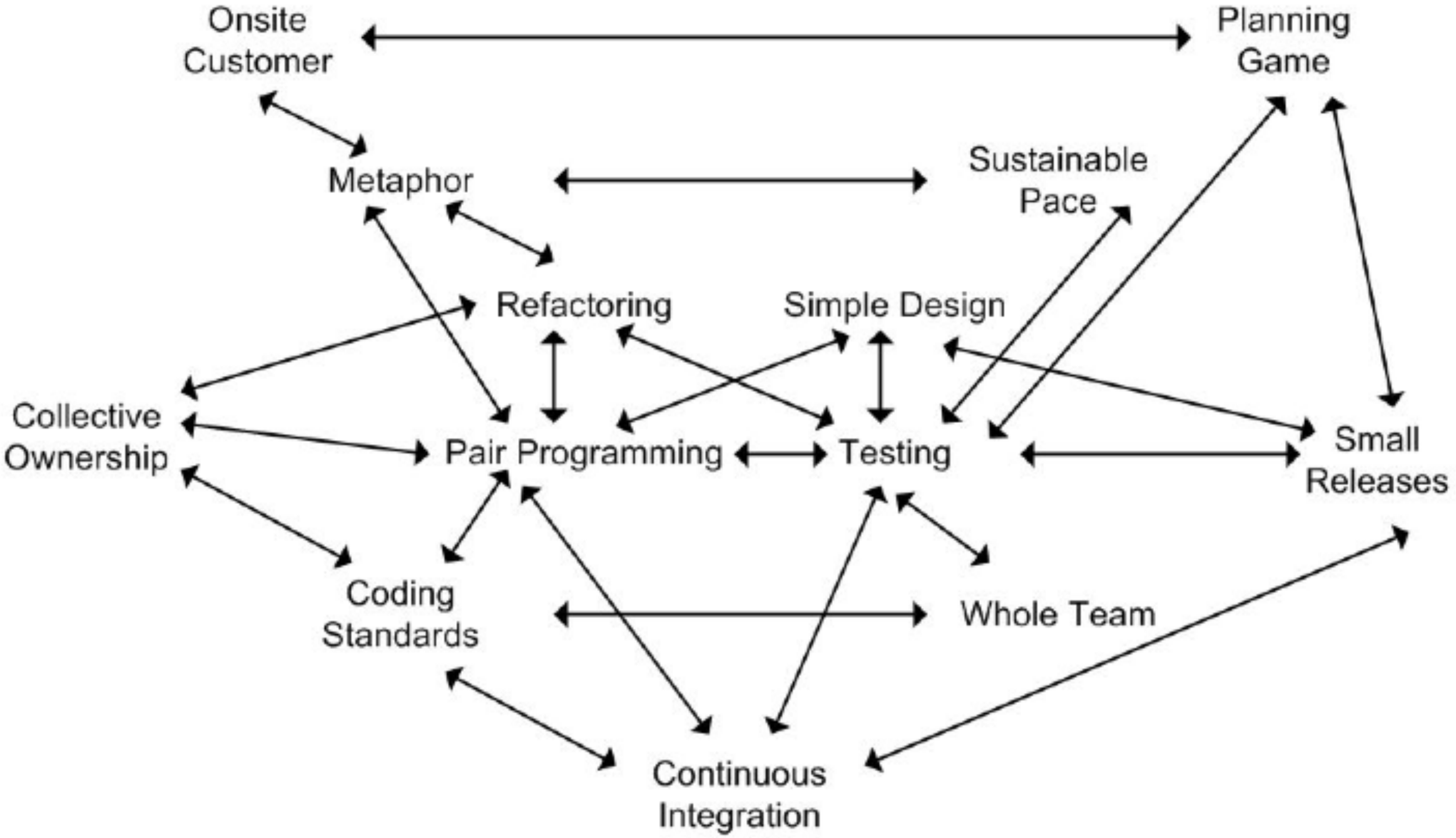
Then perhaps you could integrate after a few hours. Besides, if you didn't integrate quickly the chance of conflict rises and the cost of integration goes up steeply.

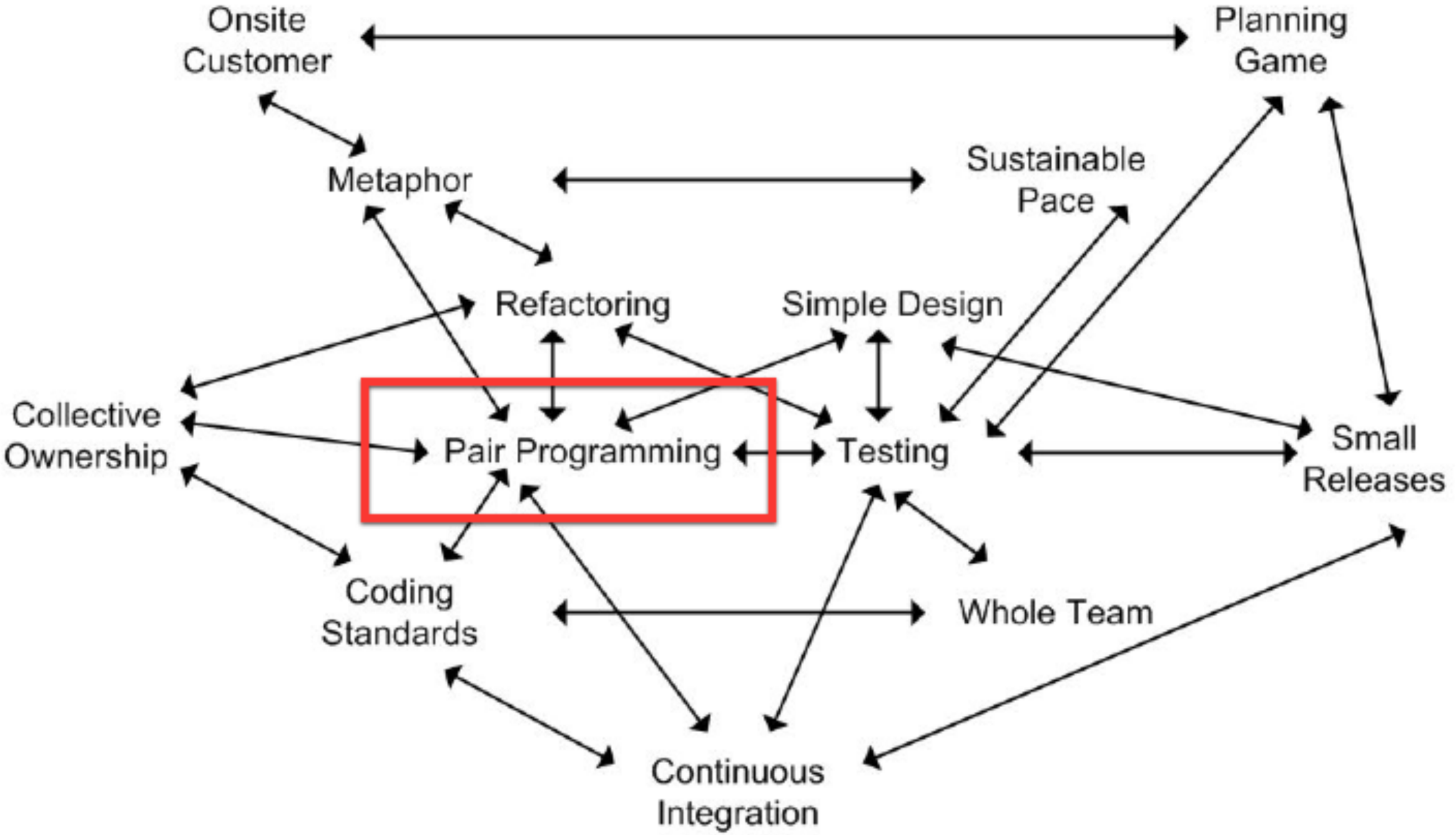
Any one practice doesn't stand well on its own. They require the other practices to keep them in balance.

— *Kent Beck*

The individual pieces are simple. The richness comes from the interactions of the parts.

— *Kent Beck*

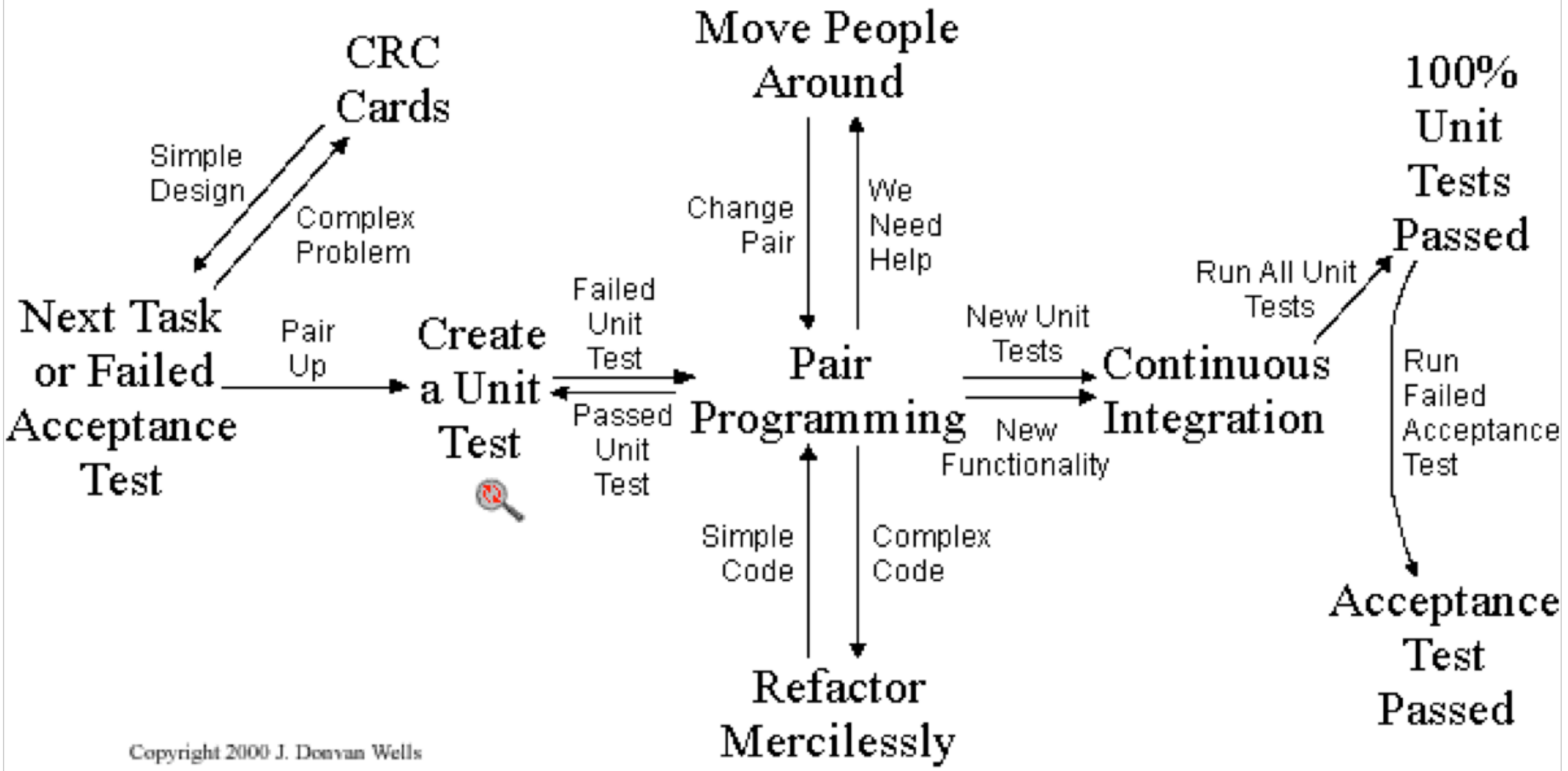

Onsite Customer
Planning Game
Metaphor
Sustainable Pace
Refactoring
Simple Design
Collective Ownership
Pair Programming
Testing
Small Releases
Coding Standards
Whole Team
Continuous Integration


Onsite Customer
Planning Game
Metaphor
Sustainable Pace
Refactoring
Simple Design
Collective Ownership
Pair Programming
Testing
Small Releases
Coding Standards
Whole Team
Continuous Integration

Collective Code Ownership

Extreme Programming

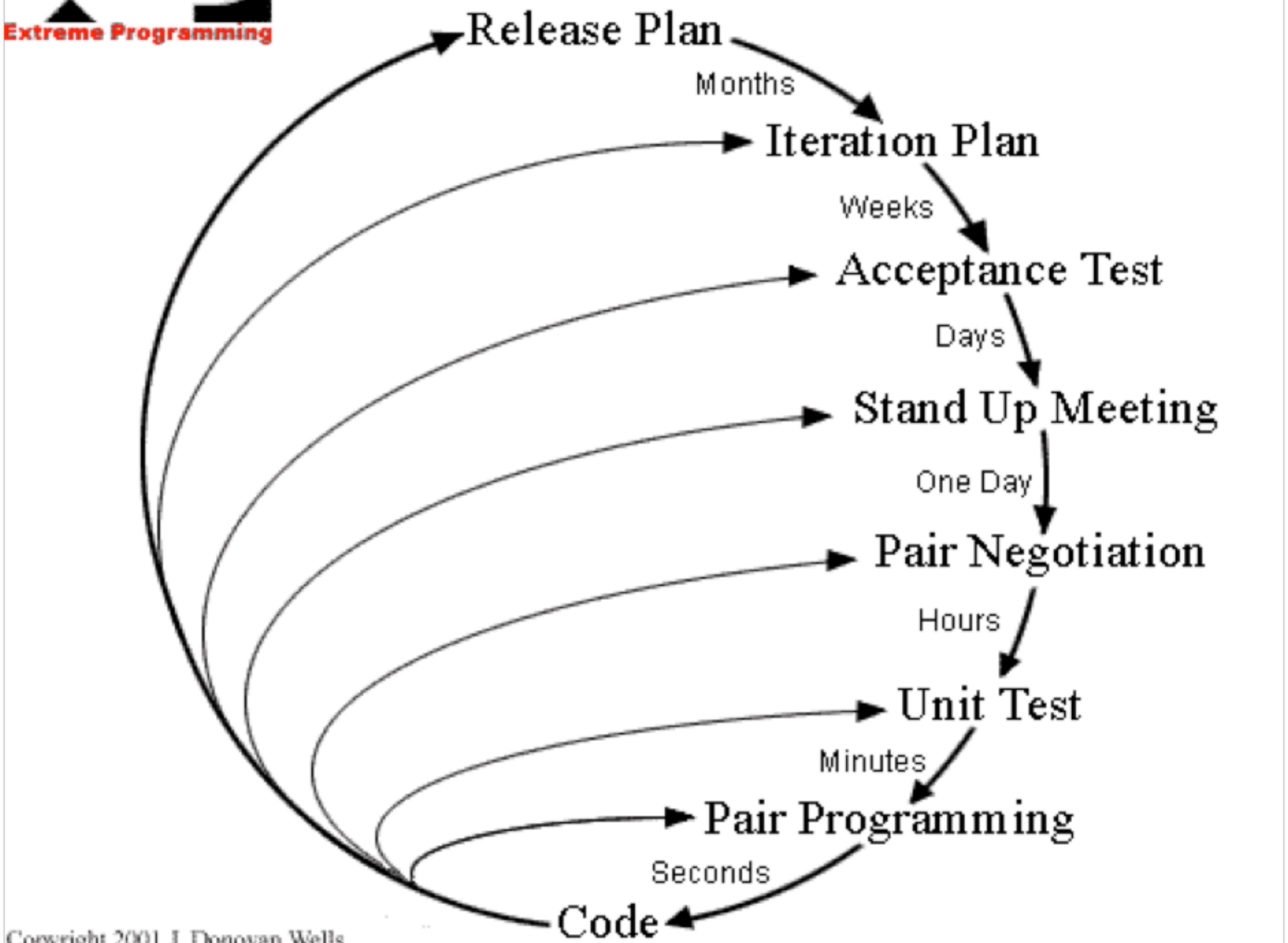
Zoom Out



Extreme Programming

Planning/Feedback Loops

Zoom Out



Why we sometimes find it hard to start pairing

I feel intimidated to work with someone more experienced

- *Nobody* knows what they're doing
- The impostor syndrome
  - "I'm just making stuff up as I go along"
  - "I'm going to be found out"
- The Dunning-Kruger effect
  - "There's so much that I don't know"
  - "Everyone here's a better programmer than I am"

I don't want to disturb someone to ask them to pair

- Agree explicit rules about interruption for pairing
- Fixed pair rotation times e.g. every day/half-day
- No need for same pair to work on same story from beginning to end

No other developer available to pair e.g. stuck in meetings

- Core pairing hours
- Doing more pairing ought to reduce the need for as many meetings
- Shorter more efficient meetings / make attendance voluntary
- Spike on something / refactor something

I found my last pairing session very "painful"

- Be patient - pairing is a skill which takes time to learn
- Decide to try some a new tactic
- Pair with someone different

Why we sometimes find the process of pairing difficult

- Exhaustion / fatigue
  - Take regular breaks
- Difficult to articulate thought processes
  - Sketching on a whiteboard / piece of paper
  - Pseudo code sketching
- No time for personal admin
  - Don't try to pair all day

- Some types of task are hard to pair on e.g. writing commit notes/ documentation
    - Allow a single person to write and have the pair review it at the end
- One person hogs/avoids the keyboard
    - Pairing ping-pong
- One person is distracted by email/IM notifications
    - Switch off all distractions
- Differences in development environment e.g. editor, shell, OS

Q&A

References

- http://www.extremeprogramming.org/
- http://codon.com/i-have-no-idea-what-im-doing
- http://en.wikipedia.org/wiki/Impostor_syndrome
- http://en.wikipedia.org/wiki/Dunning%E2%80%93Kruger_effect