Extreme Programming:

“Once over lightly”

Andrew Black

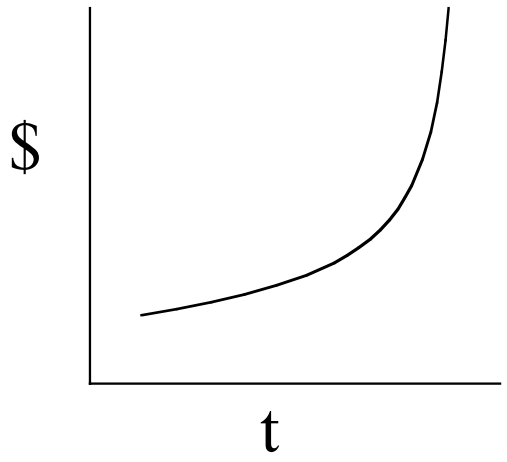
Presentation based on material from Nick Southwell, WilliamWake, and others

XP is...

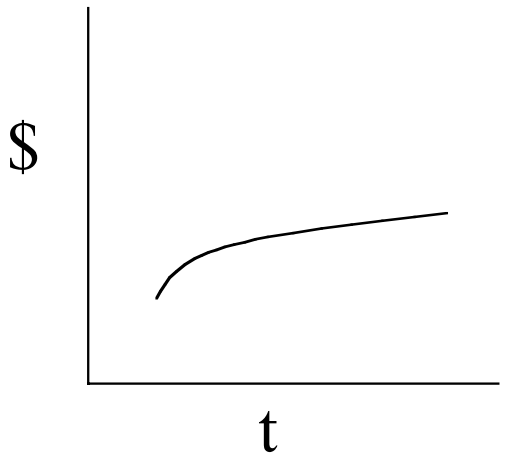
A lightweight development methodology that emphasizes:

- ongoing user involvement
- testing
- pay-as-you-go design

Background: Cost of Changes

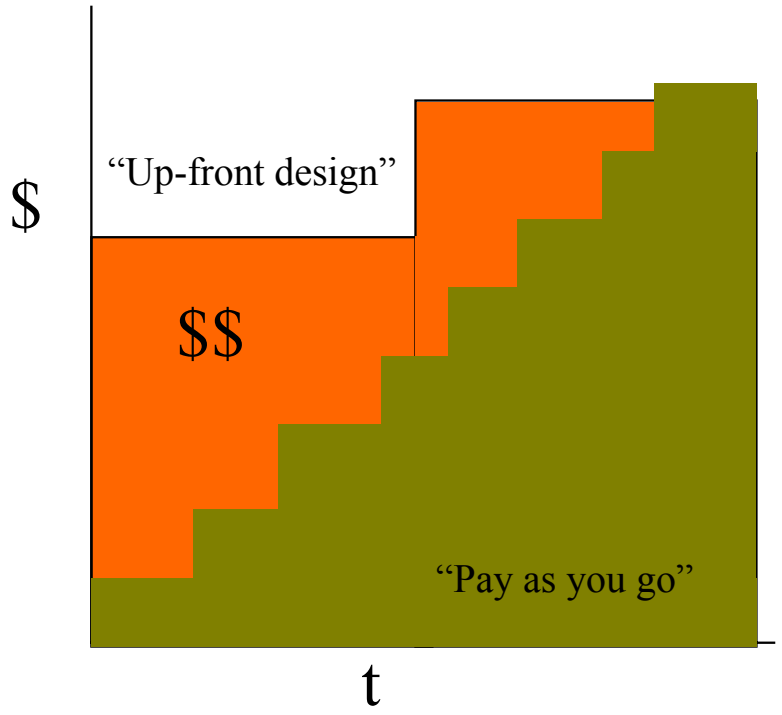


"Then" (exponential)



"Now" (flattened)

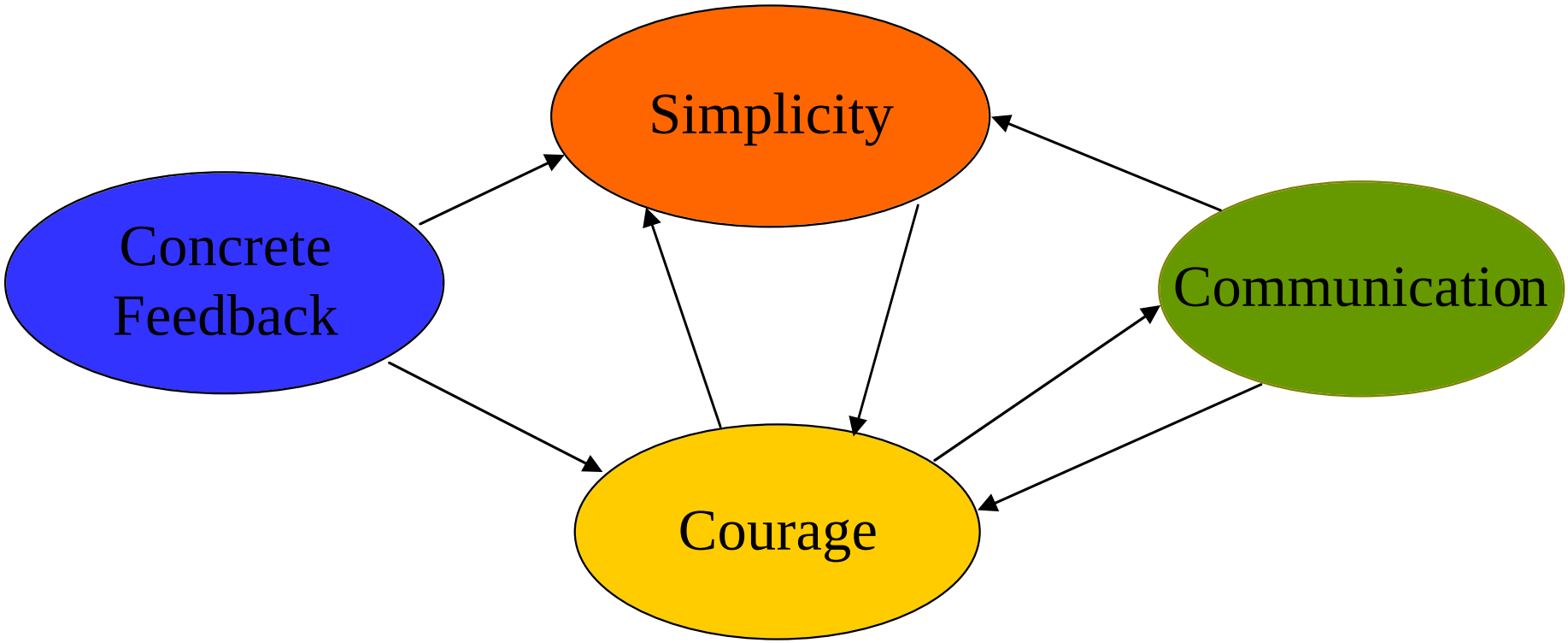
Background: Cost of Money



Key values of XP

- Communication
  - Problems with projects can invariably be traced back to somebody not talking to somebody else.
- Simplicity
  - It is better to do a simple thing today, and pay a little more tomorrow, than to do a complicated thing today that may never be used.
- Concrete Feedback
  - Feedback at all time scales keeps the project on track.
- Courage
  - Together with the first three values, Courage allows you to make high-risk, high-reward experiments. Without them, it's just hacking.

XP values



XP Principles

- Get rapid feedback
- Assume simplicity
- Incremental change
- Embrace Change
- Do quality work

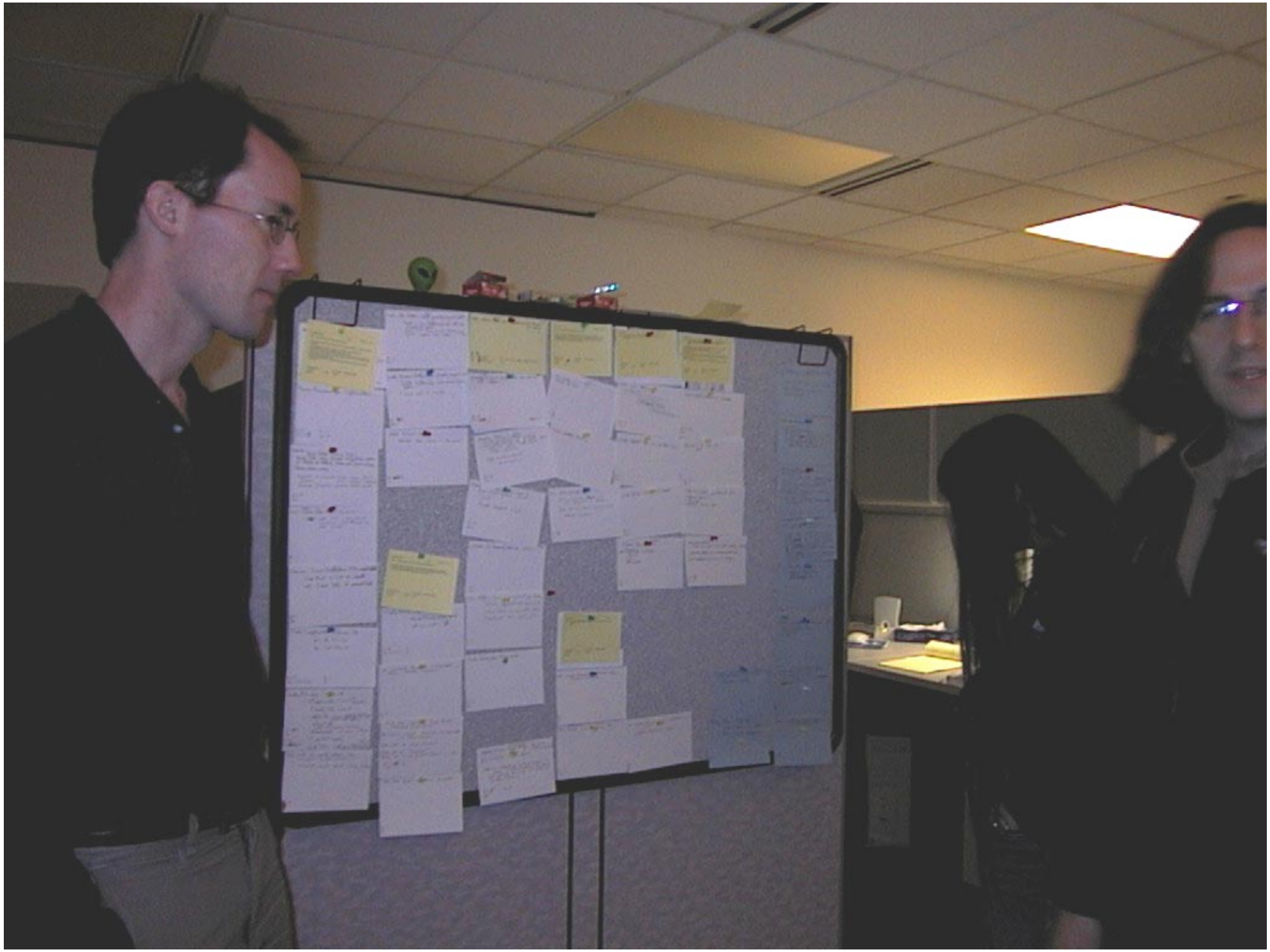
XP Practices

- Planning Game
- Metaphor
- Testing
- Refactoring
- Pair programming
- Small releases
- On-site customer
- Simple design
- Collective ownership
- Continuous integration
- 40-hour week
- Coding Standards

Planning Game

User stories = lightweight use cases

- 2-3 sentences on a file card that
  - —the customer cares about
  - —can be reasonably tested
  - —can be estimated & prioritized



Planning Game (cont.)

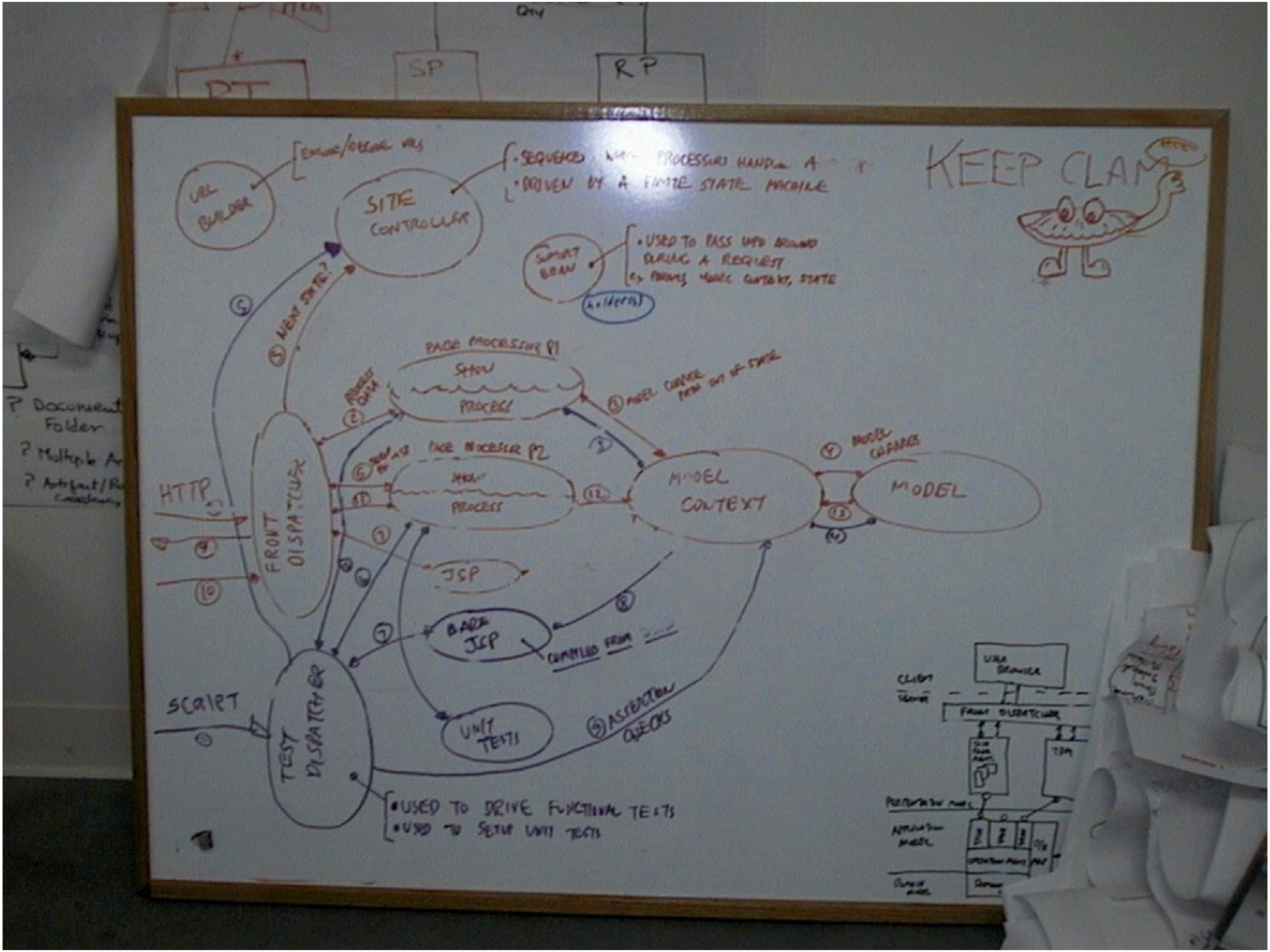
- Users write stories
- Developers estimate them
- Users split, merge, & prioritize
- Plan overall release (loosely) and the next iteration
  - Don’t plan too far ahead

Small Releases

- Make every release as small as possible
  - Release makes sense as a whole
- Make simple designs, sufficient for the current release
- Small releases provide:
  - rapid feedback
  - sense of accomplishment
  - reduced risk
  - customer confidence
  - adjustments to changing requirements

Metaphor

- Guide the project with a single Metaphor
  - *e.g.*, the UI is a desktop
- Must represent the architecture
  - makes it easier to discuss
- The customer must be comfortable with it


SP
RP
VAL BUILDER
SITE CONTROLLER
KEEP CLAM
HTTP
FRONT DISPATCHER
PAGE PROCESSOR P1
SHOW PROCESS
PAGE PROCESSOR P2
SHOW PROCESS
MODEL CONTEXT
MODEL
JSP
BARE JSP
SCRIPT
TEST DISPATCHER
UNIT TESTS
USED TO DRIVE FUNCTIONAL TESTS
USED TO SETUP UNIT TESTS
USER BROWSER
TPM

Simple Design

- The right design for software:
  - Runs all the tests.
  - Has no duplicated logic (DRY principle)
  - States every intention important to the programmers.
  - Has the fewest possible classes and methods
- Don’t worry about having to change a design later

Testing

- XP tests everything that might possibly break, all the time
- The tests *are* the specification:
  - An *executable* specification
- Two kinds of tests:
  - Functional Tests
  - Unit Tests

Functional Tests

- Specified by the user
- Implemented by users, developers, and/or test team
- Automated
- Run at least daily
- Part of the specification

Unit Tests

- Written by developers
- Written before and after coding
- Always run at 100%
- Support design, coding, refactoring, and quality.

Pair Programming

- Role of one partner
  - uses the mouse and the keyboard
  - thinks about the best way implementing the method
- Role of the other
  - is the approach going to work
  - think about test cases
  - can it be done simpler
- Pairing is dynamic
- Pairing provides discipline
- Pairing spreads knowledge about the system


ASIX


Quantex

Collective Code Ownership

- Anybody can add to any portion of the code
  - subject to current requirements
  - subject to simple design
- Unit tests protect the system functionality
- Whoever find a problem, solves it
- Everybody is responsible for the whole system

Continuous Integration

- Integration of tested code every few hours (max. a day)
- All unit tests need to run successfully
- If a test fails the pair has to repair it
- If you can’t repair it, throw away the code and start again

40 Hour Week

- If you can’t do your work in 40 hours, then you have too much work
- 40-Hour weeks keeps you fresh to tackle problems
- It prevents making silly, hard to find mistakes late at night
- Frequent planning prevents you from having too much work
- Overtime is a symptom of a serious problem

On-site Customer

- Writes functional tests
- Makes priority and scope decisions for the programmers
- Answers questions
- Does his or her own work

If you can’t get an On-site Customer, maybe the project isn’t important enough?


PRINGLES


Quantex

Coding Standards

- Complicated constructions are not allowed
  - let’s keep things simple
- Code looks uniform
  - easier to read
- No need to reformat the code
  - no ‘curly brackets wars’

Design

- Pay as you go
- Re-design when necessary
- “You aren’t gonna need it”
- “Simplest thing that could possibly work”
- “Once And Only Once”

Refactoring

- Refactor = to improve the structure of code without affecting its external behavior
- Done in small steps
- Supported by unit tests, simple design, and pair programming
- Seek “once and only once”
- Refactoring in pairs gives you more courage and confidence

Refactoring Example

Replace Magic Number by Constant:

```
return 32.5 *
    miles_traveled;
```

```
static final double
   MILEAGE_RATE = 32.5;
…
return MILEAGE_RATE *
    miles_traveled;
```

Separate Query from Modifier:

```
Stack:
    Object getTopAndPop();
```

```
    Object getTop();
    void pop();
```

Planning XP

Why Plan?

- To do the most important thing
- To coordinate with others
- To be able to respond to the unexpected

The Balance of Power

Business people make business decisions

- dates
- scope
- priority

Developers make development decisions

- estimates

Two kinds of planning:

- Release Planning
- Iteration planning

Release Planning

- Customers write stories
- Developers estimate them
  - stories that are too complex to estimate go back to the customer to be split
- Customer prioritizes the stories and fills a three week “bucket” with their choice
  - don’t worry about “dependencies”
- Do one, two (or even three?) releases like this

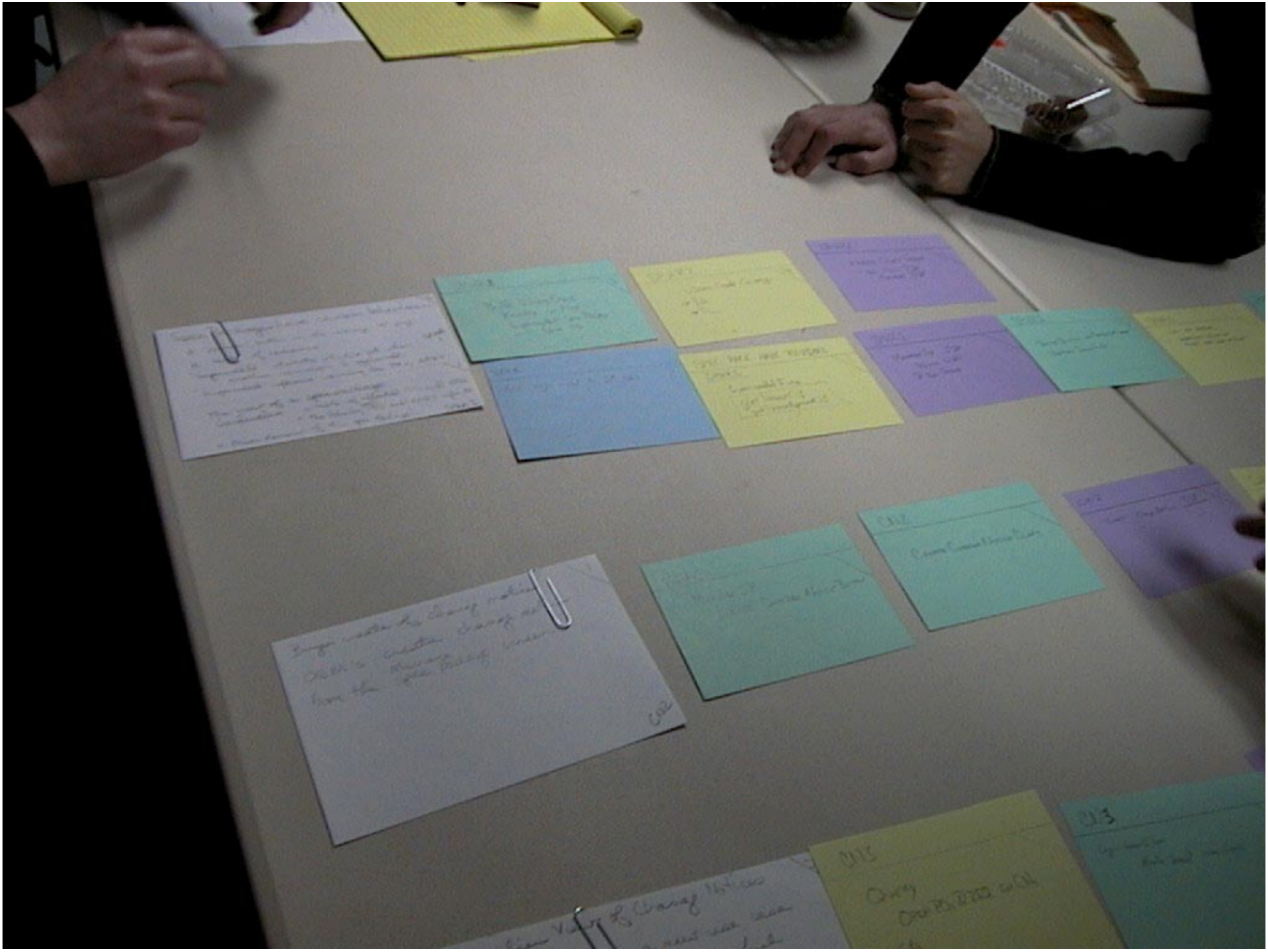
Iteration Planning

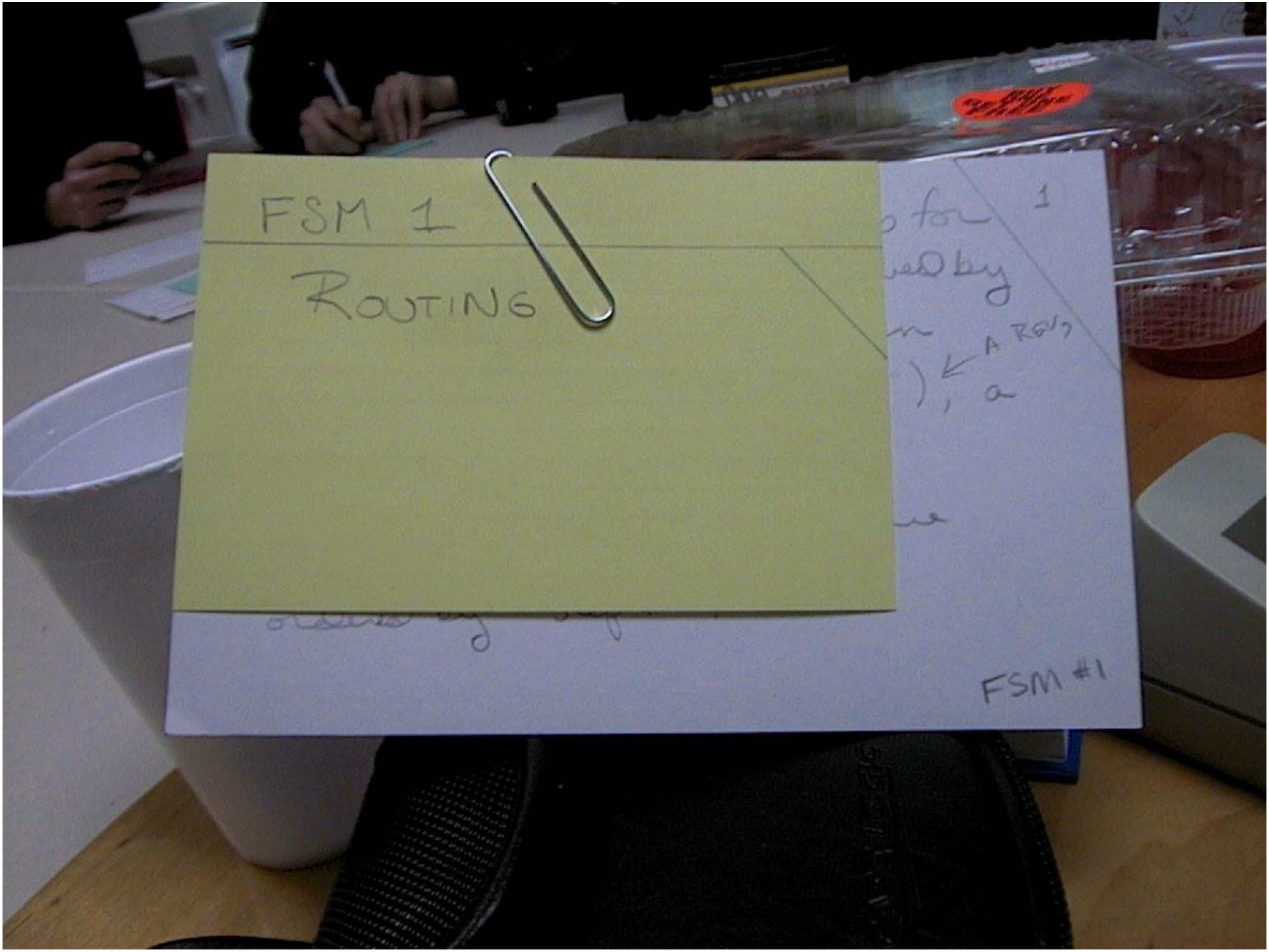
For the current release:

- Developers split each story into tasks
- Estimate the tasks collectively
- Individuals sign up to do the tasks

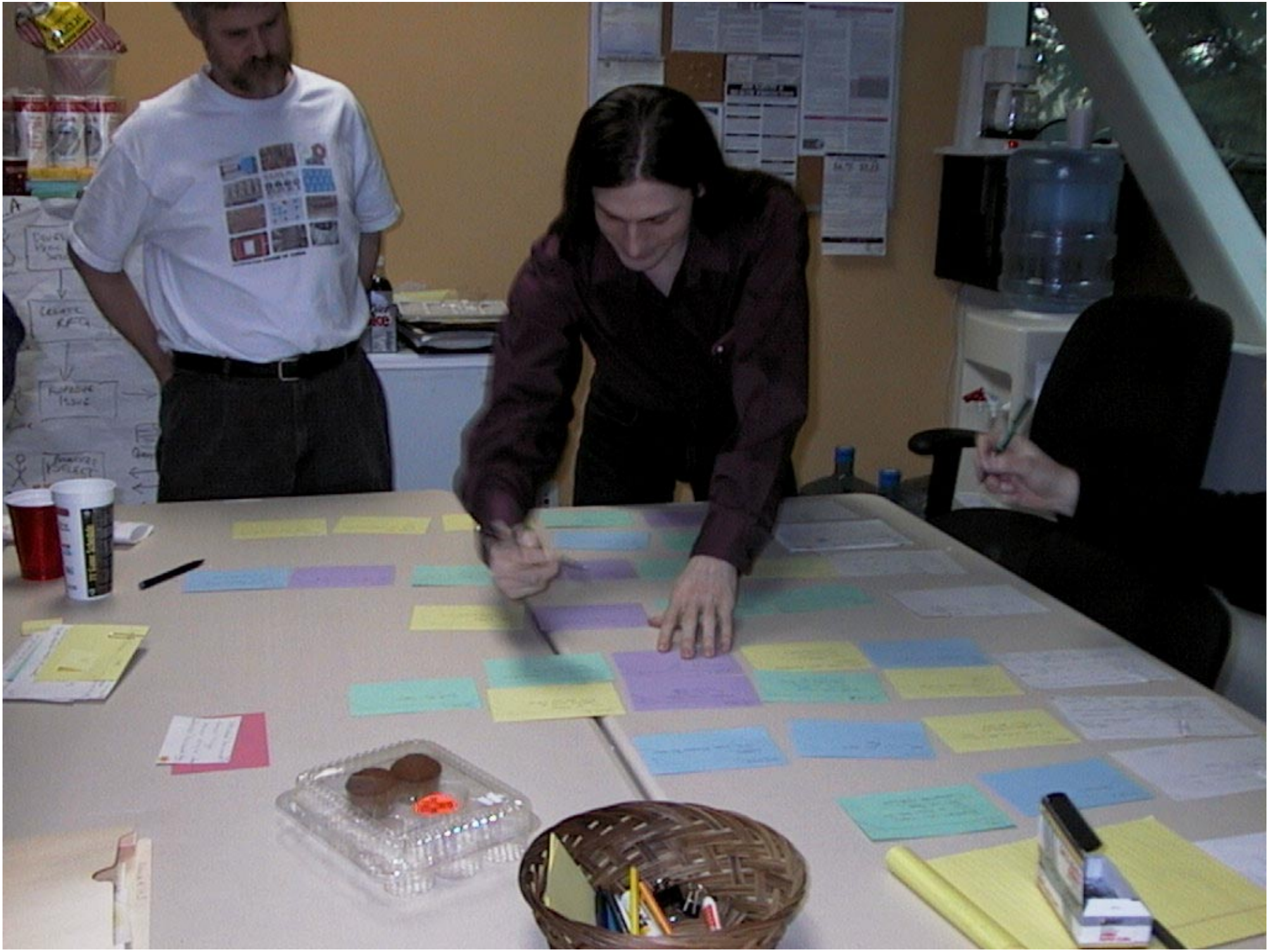
Which tasks do you do first?

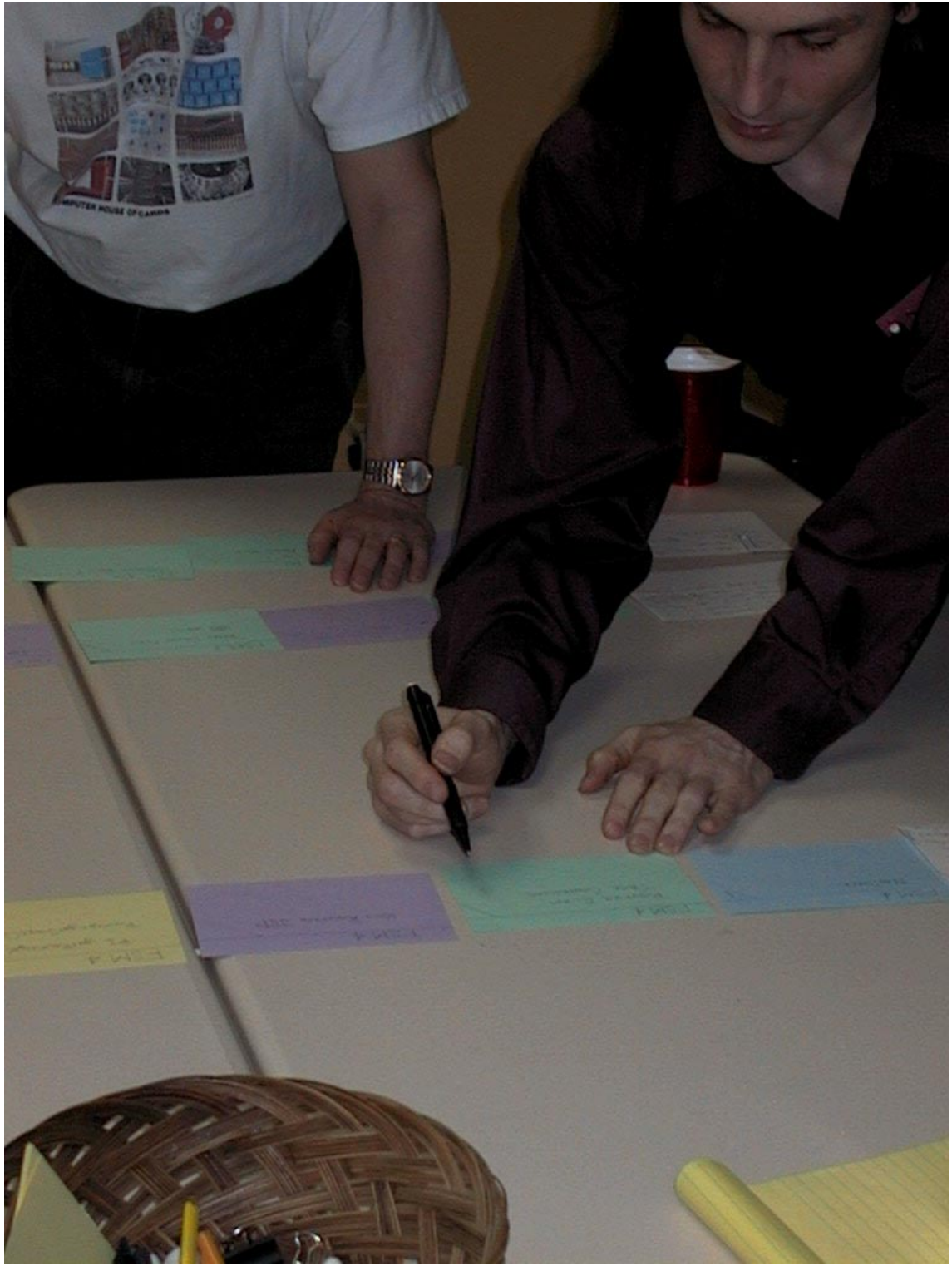
- The riskiest ones!




FSM 1
ROUTING
FSM #1


CLAMS
ENGINEERING
TASKS
MODEL
SESSION
DATABASE
PERSISTENCE
GUI
UNPLANNED
D

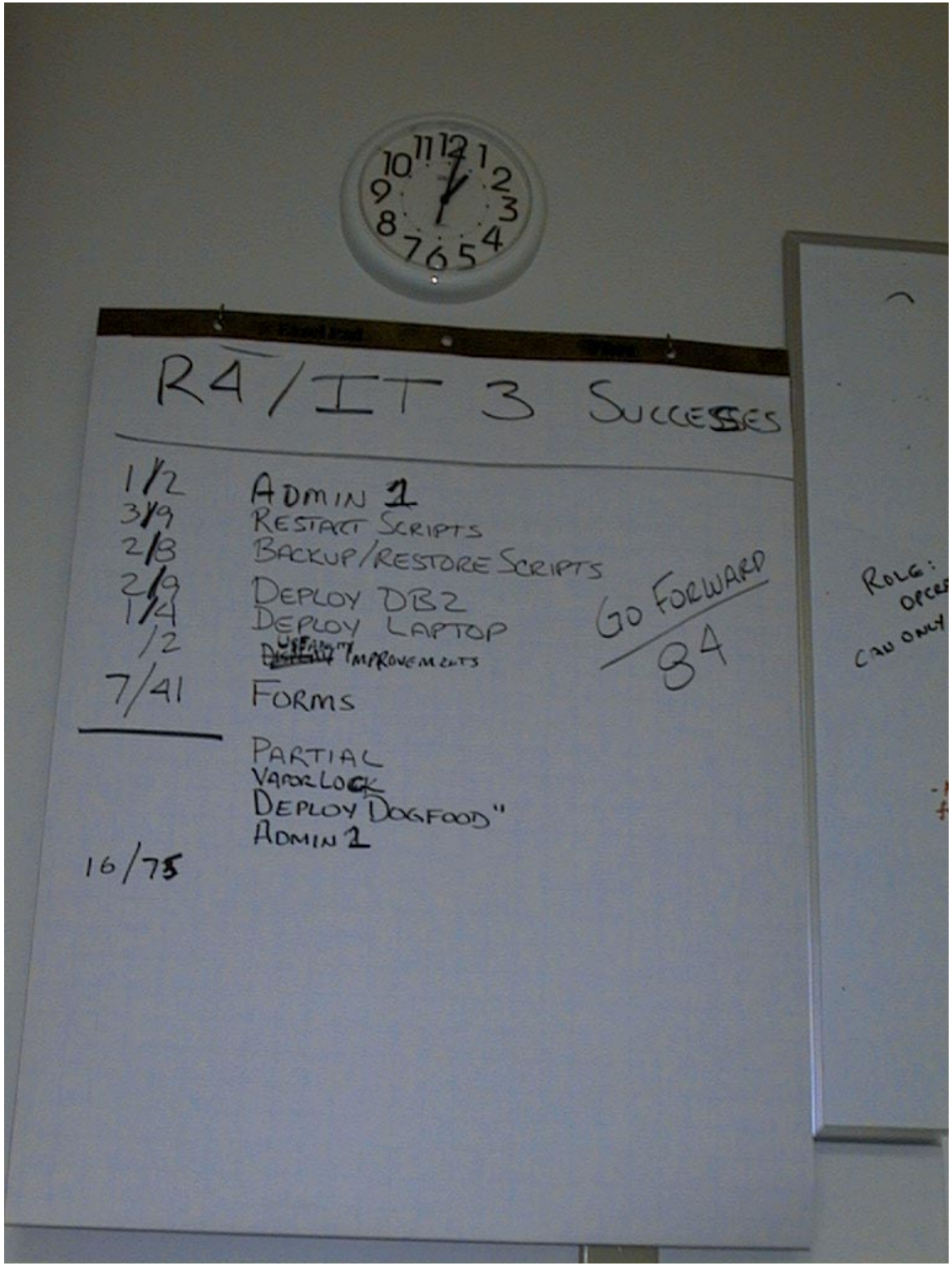


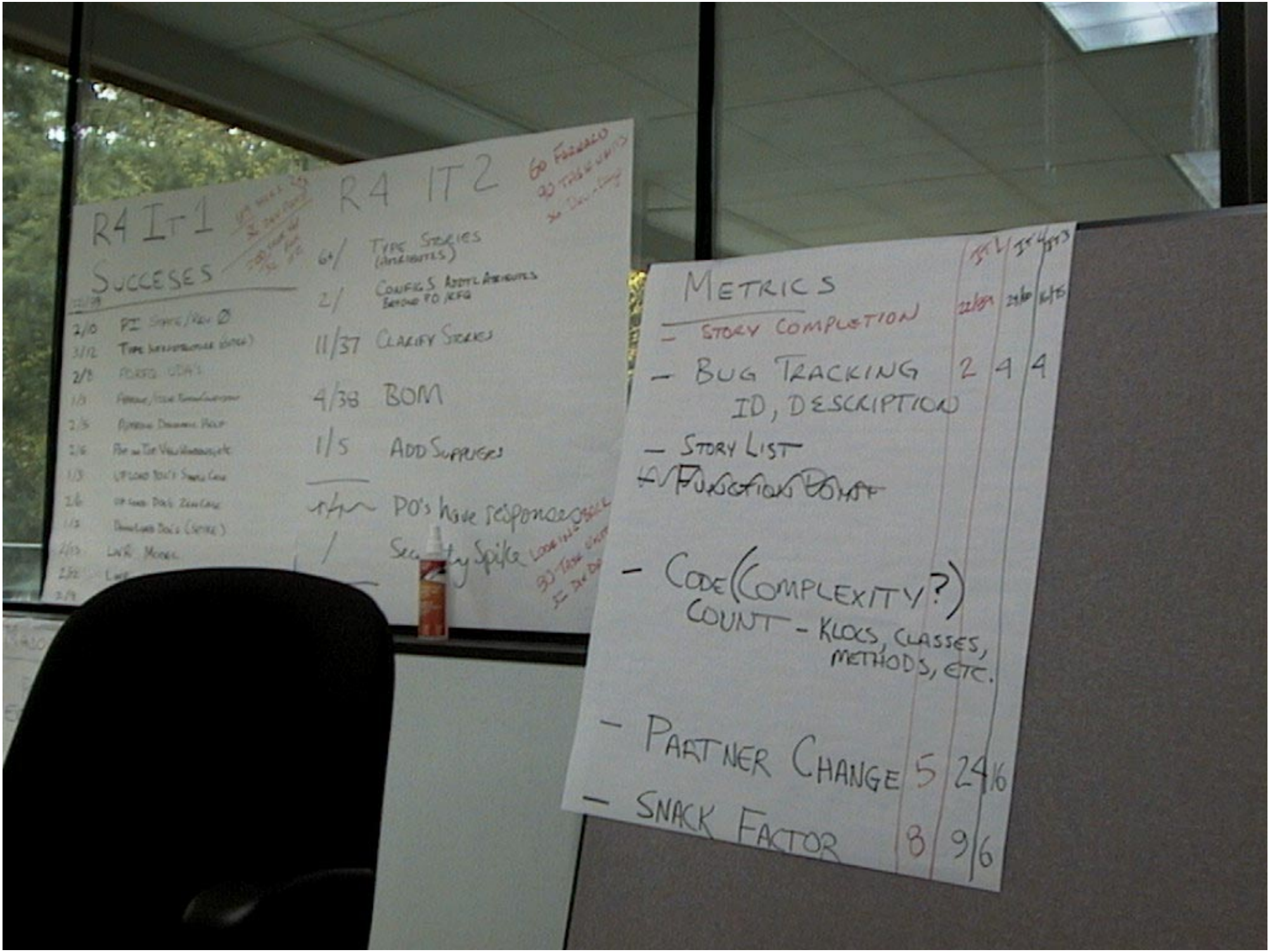


Measure what you do

It’s OK to try almost anything on an XP project, as long as you learn from it!

- an XP team is a learning organization
- it must constantly compare its performance against its estimates
- if a practice helps, do more of it.
- if a practice hinders, do less of it


R4/IT 3 SUCCESSES
1/2 ADMIN 1
3/9 RESTART SCRIPTS
2/3 BACKUP/RESTORE SCRIPTS
2/9 DEPLOY DB2
1/4 DEPLOY LAPTOP
/2 USEABILITY IMPROVEMENTS
7/41 FORMS
PARTIAL
VAPORLOCK
DEPLOY DOGFOOD"
ADMIN 2
16/75
GO FORWARD
84
ROLE:
CAN ONLY

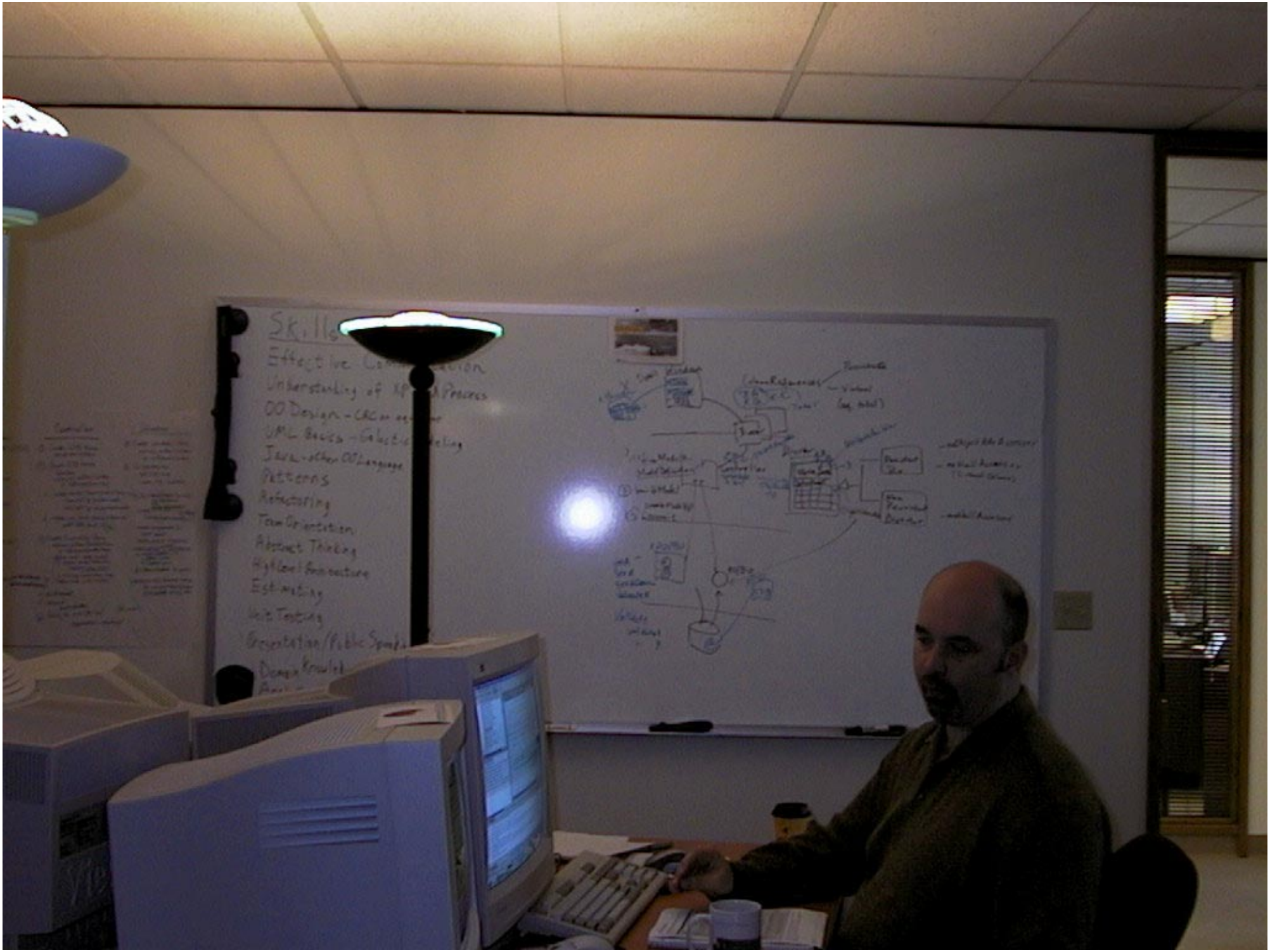

R4 IT1
SUCCESES
R4 IT2
6/ TYPE STORIES (ATTRIBUTES)
2/ CONFIGS ADDTL ATTRIBUTES BEYOND PO KEG
11/37 CLARIFY STORIES
4/38 BOM
1/5 ADD SUPPLIERS
PO's have response
Security Spike
METRICS
- STORY COMPLETION
- BUG TRACKING ID, DESCRIPTION 2 4 4
- STORY LIST
- CODE (COMPLEXITY?) COUNT - KLOCS, CLASSES, METHODS, ETC.
- PARTNER CHANGE 5 2 4/6
- SNACK FACTOR 8 9/6

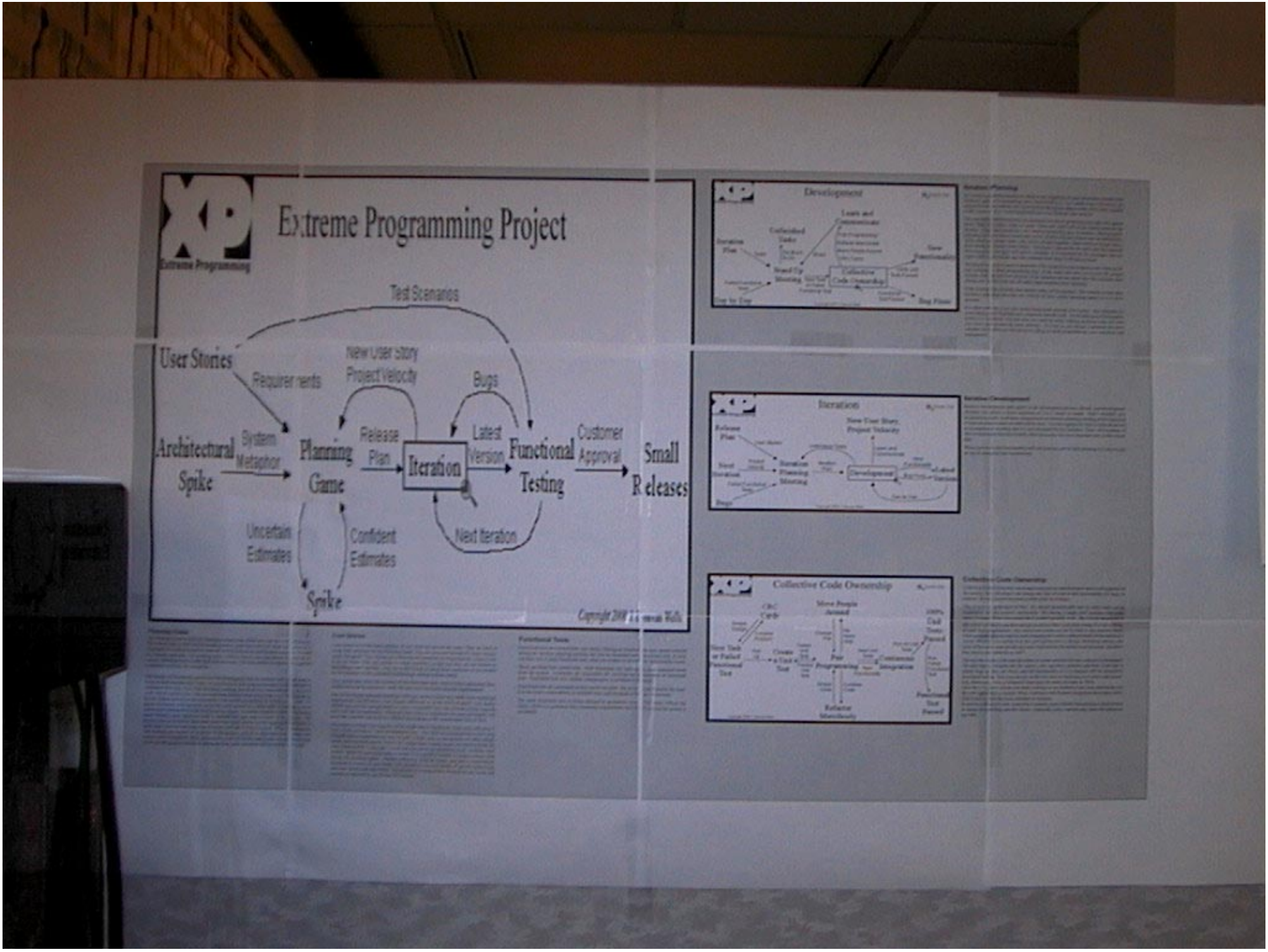
1. Process Discussions (All Hands)
2. Create Supplemental Ev. Dtl
3. Create Supp. Ev. Relat'ship
4. Add Values Grid
5. Funds on Party tab
6. Fin Acct. Brz
7. Fin Acct Dtl
8. APP Brz

Auto buttons
Copy Cell
Prop highlight
Dup Records
Save Changes

Adopting XP

- Some practices can be done solo, others by team, others require users to help.
  - —Customer involvement
  - —Functional tests and unit tests
  - —Simple design & refactoring
  - —Pair programming


Skills
Effective
Understanding of
OO Design
UML Basics
Java
Patterns
Refactoring
Team Orientation
Abstract Thinking
Estimating
Unit Testing


Extreme Programming Project
Extreme Programming
Test Scenarios
User Stories
Requirements
New User Story
Project Velocity
Bugs
Architectural Spike
System Metaphor
Planning Game
Release Plan
Iteration
Latest Version
Functional Testing
Customer Approval
Small Releases
Uncertain Estimates
Confident Estimates
Spike
Next Iteration
Development
Iteration
Collective Code Ownership

Nick Southwell (Motorola Personal Networks) asks: Can We Use XP?

- XP is for small teams
  - XP relies on verbal communication instead of formal documentation
- XP is for “greenfield” as opposed to “legacy” projects
  - We have lots of code with no tests, or documentation
  - We have no coding standards
  - Many parts of the system are understood by only one person
- XP requires leadership, discipline and team buy-in
  - All the team must believe that XP can work
  - There are no shortcuts
  - Need a leader to drive XP

Pretty Adventuresome Programming (PAP)

- About as much excitement as you're going to want
- Dials up pretty high: 9.3 or so.
- Wow that XP is neat! We almost do it too!

See Alistair Cockburn at
http://c2.com/cgi/wiki?PrettyAdventuresomeProgramming

Extreme Programming Requires:

- Pair Programming
- Deliver an increment every 3 weeks
- Customer on the team full-time
- Regression tests that pass 100% of the time

In return you don’t have to:

- Put comments in the code
- Write formal requirements
- Write design documents

Now, on this project we're pretty close:

- Our guys are spread around the building and the country, so we don't actually do pair programming
- Actually, we deliver our increments every 4-6 months
- We don't have customers anywhere in sight
- We don't have any unit tests

But at least:

- We don’t have many comments in the code
- We don’t have formal requirements document
- We don’t have design documents

So we're ALMOST extreme!

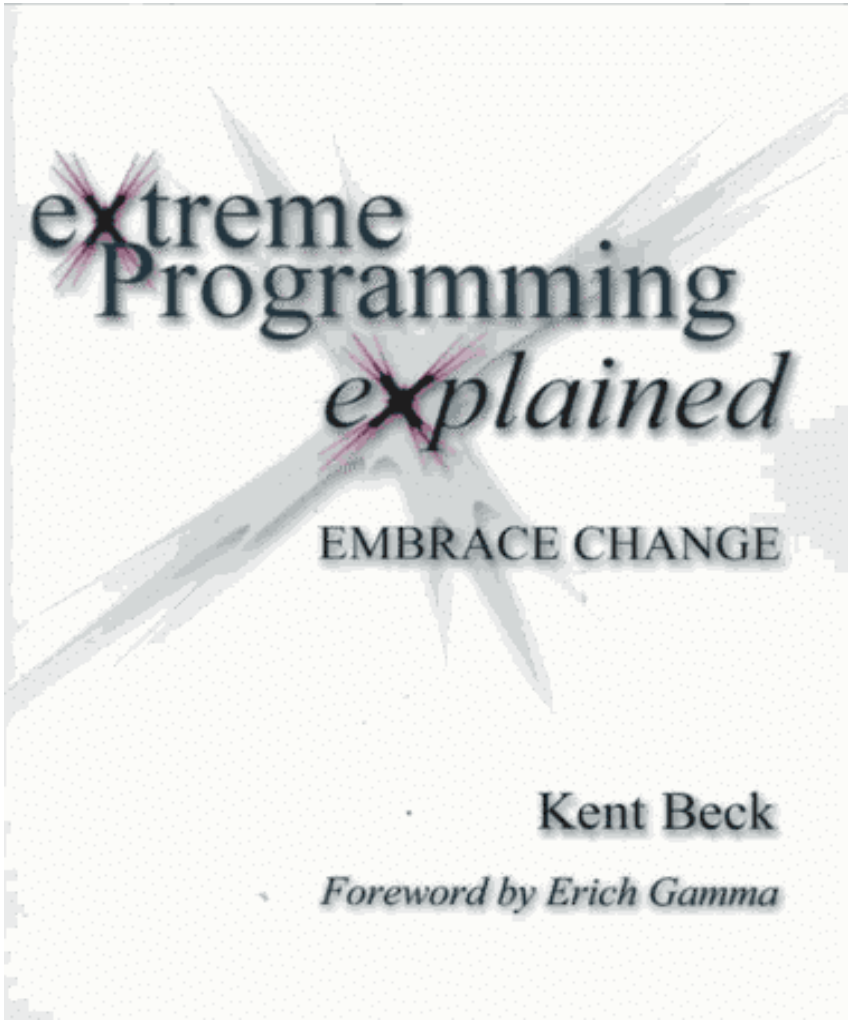
Don't use XP to legitimize not doing those things that you don't want to do, without doing the XP practices that protect you from not doing them!

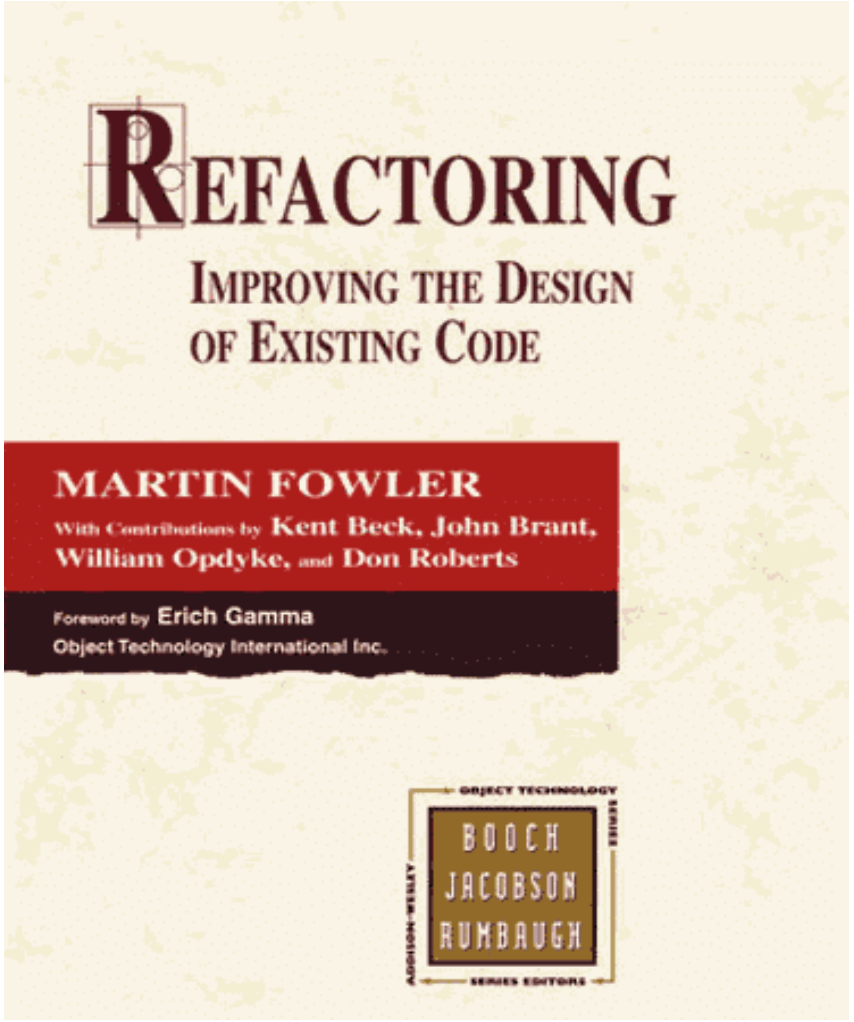
"Almost XP" = not XP at all

Internet Links to XP

- http://www.xprogramming.com
  - Ron Jeffries's site. Explains xp and offers resources for learning more.
- http://www.extremeprogramming.org/
  - Don Well's site. A great intro to XP. Presents rules and practices clearly.
- http://c2.com/cgi/wiki?ExtremeProgramming
  - The Twelve Practices of ExtremeProgramming
- http://c2.com/cgi/wiki?ExtremeProgrammingRoadmap
  - roadmap to find your way to the most important pages in a logical order.
- http://www.ObjectMentor.com/
  - Offers XP training. Based in Libertyville. Various papers on XP.
- http://www.cs.utah.edu/~lwilliam/Papers/
  - Articles on Pair Programming

Books About XP





Other Approaches

- UML: XP uses it on the whiteboard
  (if at all)
- Rational Unified Process: XP has many fewer roles & documents; XP emphasizes team over artifacts
- SCRUM: XP compatible

Summary

- XP is code centered
  - do only those things that speed up code production
  - do only those things developers like to do
    - coding and direct feedback through testing
- XP is people oriented
  - knowledge transfer through communication with real people
- XP is lightweight
  - do away with all overhead
  - create quality products by rigorously testing the code
  - only tested for small groups of developers
- The XP principles are not new