

# Agile Goa 2007

## Introduction of eXtreme Programming



Vikas Hazrati

Oct 31, 2007

# Agenda

- What is XP?
- History of Extreme Programming
- XP Core Components
- XP Values
- XP Principles
- The Whole XP Team
- XP Flowcharts
- FAQs
- Questions



# What is XP?

- XP is a philosophy of software development based on well laid out values, principles and practices.
- Goal of XP is outstanding software development at lower cost, with fewer defects, high productivity and much higher return on investment.



# What is XP?

- Another methodology but why?
  - Social Change- Giving up defences
  - Based on Excellent programming techniques, clear communication and teamwork
  - Lightweight – only do whatever adds value to the customer
  - Addresses constraints in software development
  - Can work with teams of any size
  - Adapts to rapidly changing requirements



# What is XP?

- XP addresses risks at all levels of development process
  - Schedule Slips
  - Defect Rate
  - Business misunderstood
  - Business changes
  - False feature list
  - Staff turnover

# History of Extreme Programming

- Early Influences
  - Incremental, stakeholder-driven design process from Alexander
  - Programming as learning from Papert, Kay
- Kent Beck & Ward Cunningham
  - Mid-80s – Pair programming at Tektronix
  - 80s, 90s – Smalltalk culture produces refactoring, continuous integration, constant testing, close customer involvement
  - Generalized to other environments
  - Early 90s – Core values developed within patterns community, Hillside Group

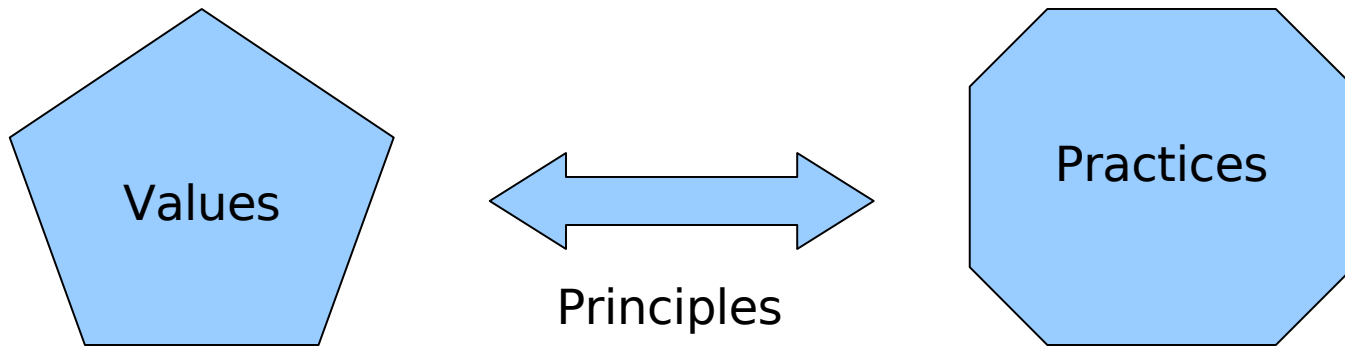


# History of Extreme Programming

- Scrum practices incorporated and adapted as planning game
- 1999 – Extreme Programming Explained
- 1999 – Fowler publishes Refactoring
- 1999 – XP Immersion held, e-group formed
- 2000 – more books, first conferences
- Evolution continues through today



# Core Components



- Values bring purpose to practices.
- Practices are evidence of values.
- Principles are domain specific guidelines for life.

# The Five Core Values of XP

- Communication.
- Simplicity.
- Feedback.
- Courage.
- Respect

# Communication

- Often problem that arise in SW project can be tracked back to lack of communication.
- XP enforces the *Communication Value* by employing many *practice* that could not be carried without communicating (e.g. pair programming, unit testing etc.).
- XP employs a *Coach* whose job is that of noticing when people are not communicating and reintroduce them.

# Simplicity

- "Do the simplest thing that could possibly work" (DTSTTCPW) principle (elsewhere known as KISS).
- An XP coach may say DTSTTCPW when he sees an XP developer doing something that is needlessly complicated.
- YAGNI principle ("You ain't gonna need it")
- Simplicity and Communication support each other mutually.

# Feedback

- Feedback works in XP at different time scales.
- Programmers have feedback on a minutes time scale on the status of the system thanks to unit tests.
- When customers write new *stories* the programmers estimate those immediately to give prompt feedback to the customer about the quality of the stories.
- The customer review the scheduler every 2-3 weeks and provide prompt feedback to the developer.

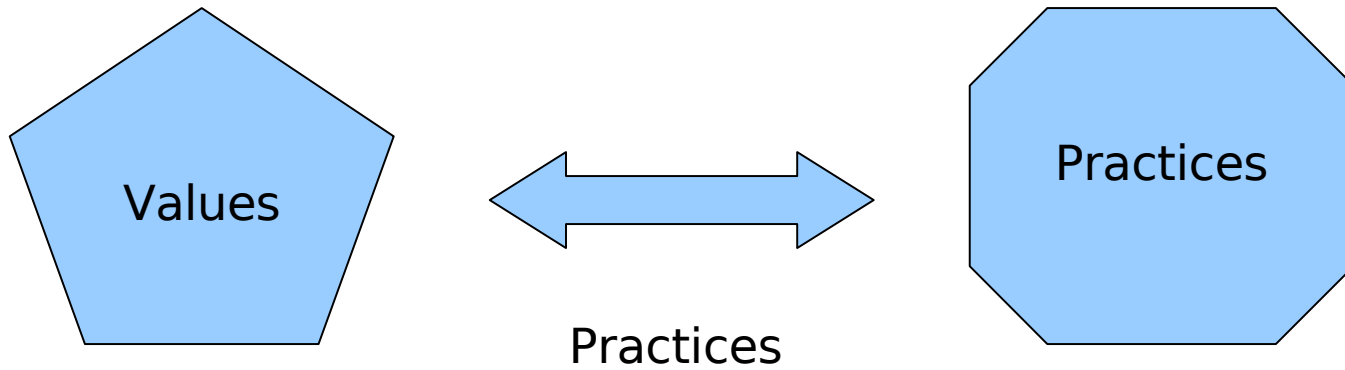
# Courage

- XP team should have the courage of throwing code away.
- XP team should have the courage of mainly refactor the architecture of the system, if architectural flaw are detected.
- Courage with counterbalancing values is dangerous. Doing something without regard for consequences is not effective teamwork.

# Respect

- Respect for team members.
- Respect for the project.

# Core Components



# XP Principles

- Humanity- People, What do people need to become good developers?
- Economics- Every action should have business value.
- Mutual Benefit- Most important and most difficult to adhere to. Extensive internal documentation.
- Self Similarity- You can copy structure of one solution to a new context. Theme, story, tests



# XP Principles

- Improvement- In software development “perfect” is a verb not adjective.
- Diversity- Teams need diversity.
- Reflection- How and Why of working.
- Flow- Steady flow of valuable software. RDITID
- Opportunities- Problems are opportunities.

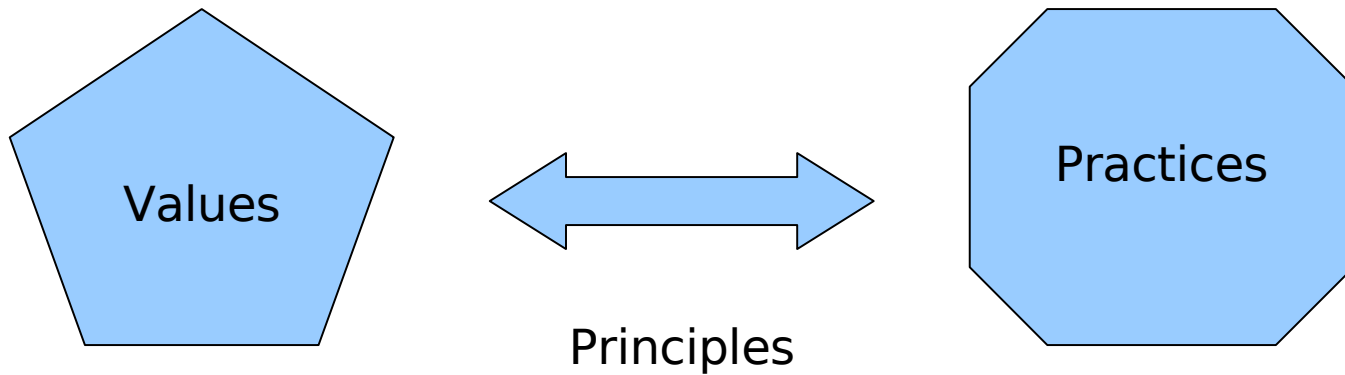


# XP Principles

- Redundancy- Do not remove redundancy that serves a valid purpose.
- Failure- Is failure a waste?
- Quality- Cost of quality? Quality ~ Productivity
- Baby Steps- Rapid small steps = leap.
- Accepted Responsibility- Responsibility cannot be assigned.



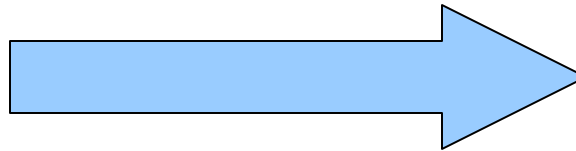
# Core Components



# Practices

Primary

Corollary



# Primary Practices

- Sit Together
- Whole Team
- Informative workspace
- Energized work
- Pair Programming
- Stories
- Weekly Cycle
- Quarterly Cycle
- Slack
- 10 minute build
- Continuous Integration
- Test First Programming
- Incremental Design



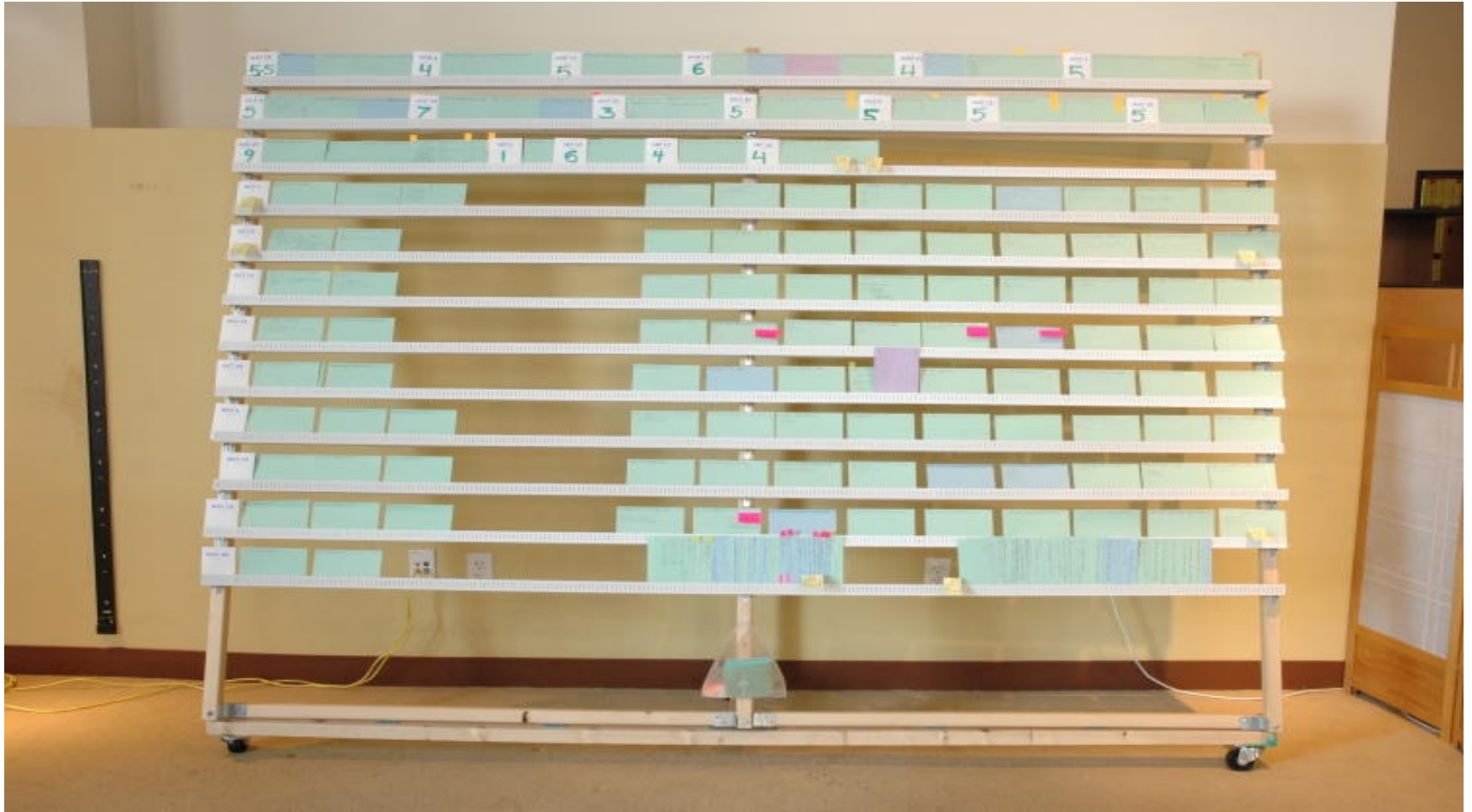
# Sit Together / Whole Team



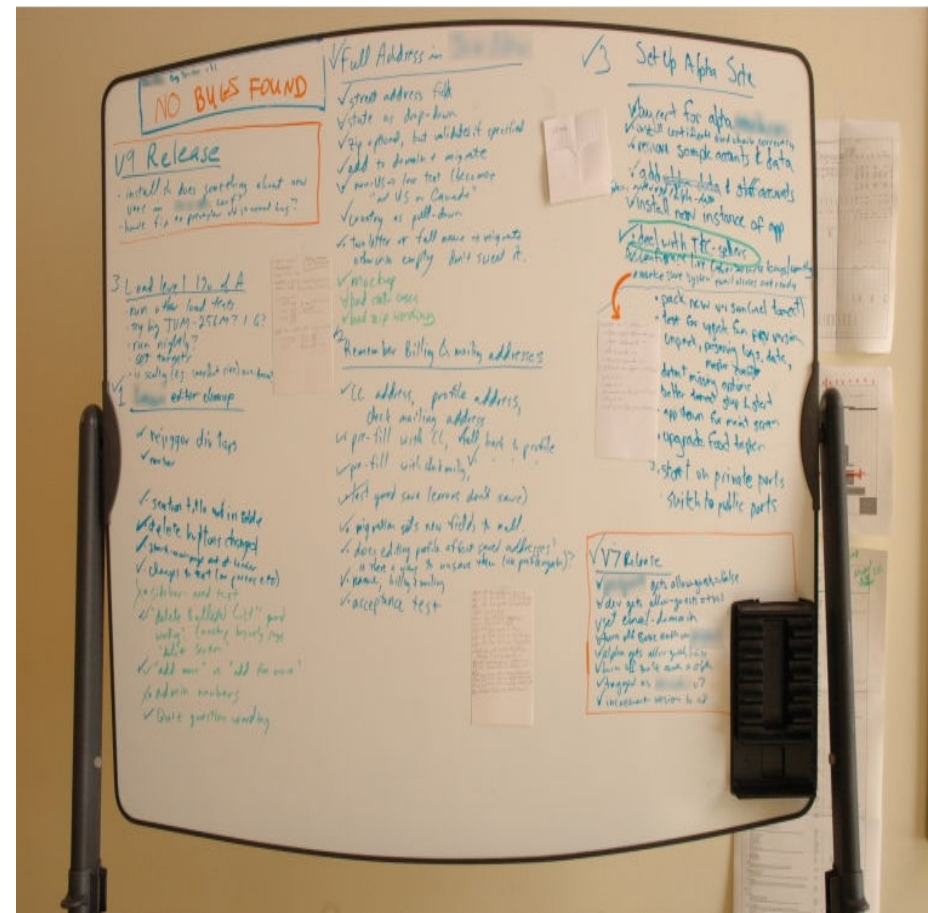
# Sit Together / Whole Team



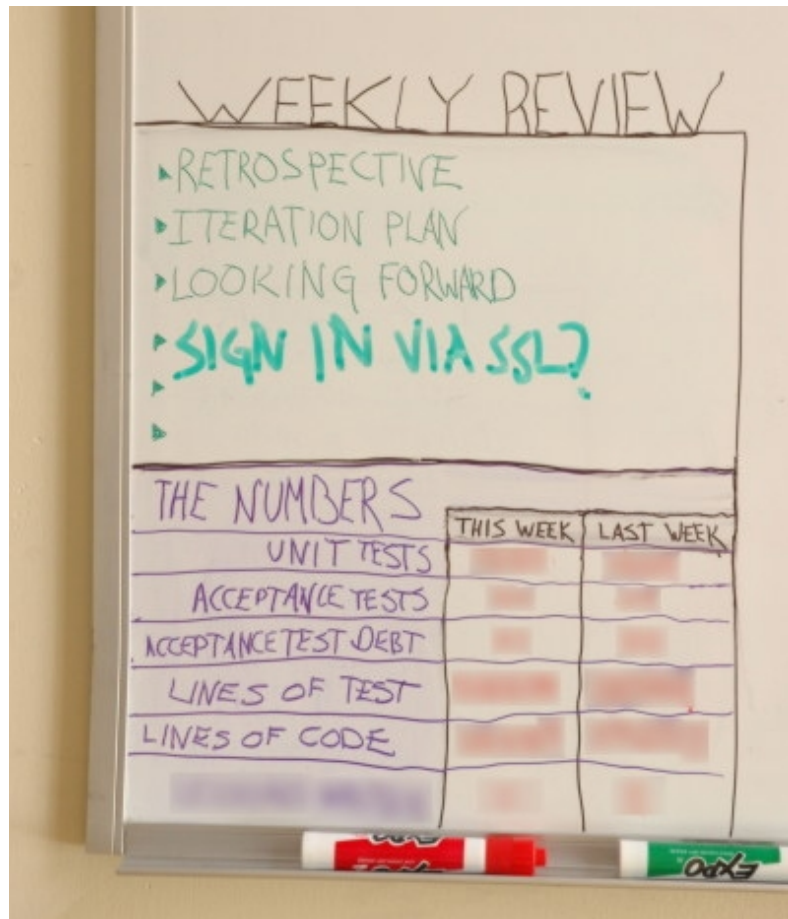
# Informative Workspace



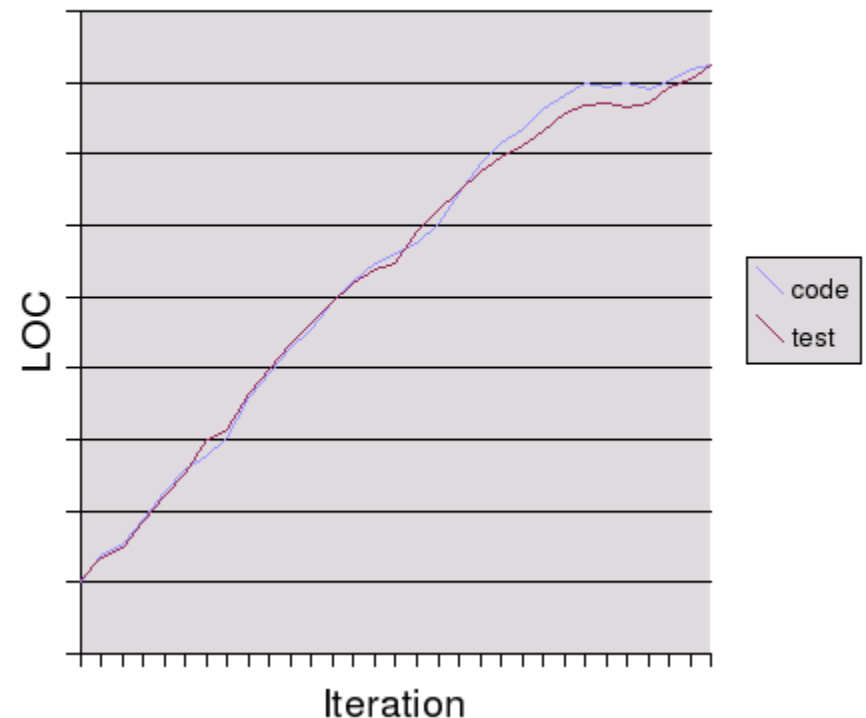
# Informative Workspace



# Informative Workspace



## Production Vs Test Code



# Energized work

- Work only as many hours as productive
- Ideally 40 hours a week
- Programming productivity is seen to be 4-5 hours a day

# Pair Programming



# Pair Programming



# Stories

- Units of customer visible functionality.
- Should be on the story wall for everyone to look at.



# Stories

## ● Stories v/s Use Cases

- A story is not a use case, a use case is not a story
- A use case defines functional requirements in total
- Set of use cases define breadth and depth of system behaviour, augmented with non-functional requirements
- A story points to a portion of breadth and depth
- Stories incrementally build up full scope
- Usually 2-3 days of work
- Further broken down into tasks

# Weekly / Quarterly Cycles

- Weekly Cycles- Start week by writing automated tests and then the week implementing them, mainly about stories and tasks.
- Quarterly Cycles- Plan for releases which cover themes. Themes can be broken down into stories for weekly cycles

# More...

- Slack
- Ten Minute Build
- Continuous Integration
- Test First Programming
- Incremental Design
  - Do the simplest thing that  
can possibly work*
  - Refactor*



# Practices

Primary

Corollary



# Corollary Practices

- Real Customer Involvement- No customer or proxy customer are not as effective
- Incremental Deployment- Build little pieces and deploy it!
- Team Continuity- Do not shuffle the team too often.
- Shrinking Teams



# Corollary Practices

- Root Cause Analysis- 5 Why's
- Shared Code- No ownership
- Code and Tests- The only artifacts
- Single code base- Avoid multiple streams
- Daily Deployment
- Negotiated Scope Contract- Ongoing negotiation on precise scope.

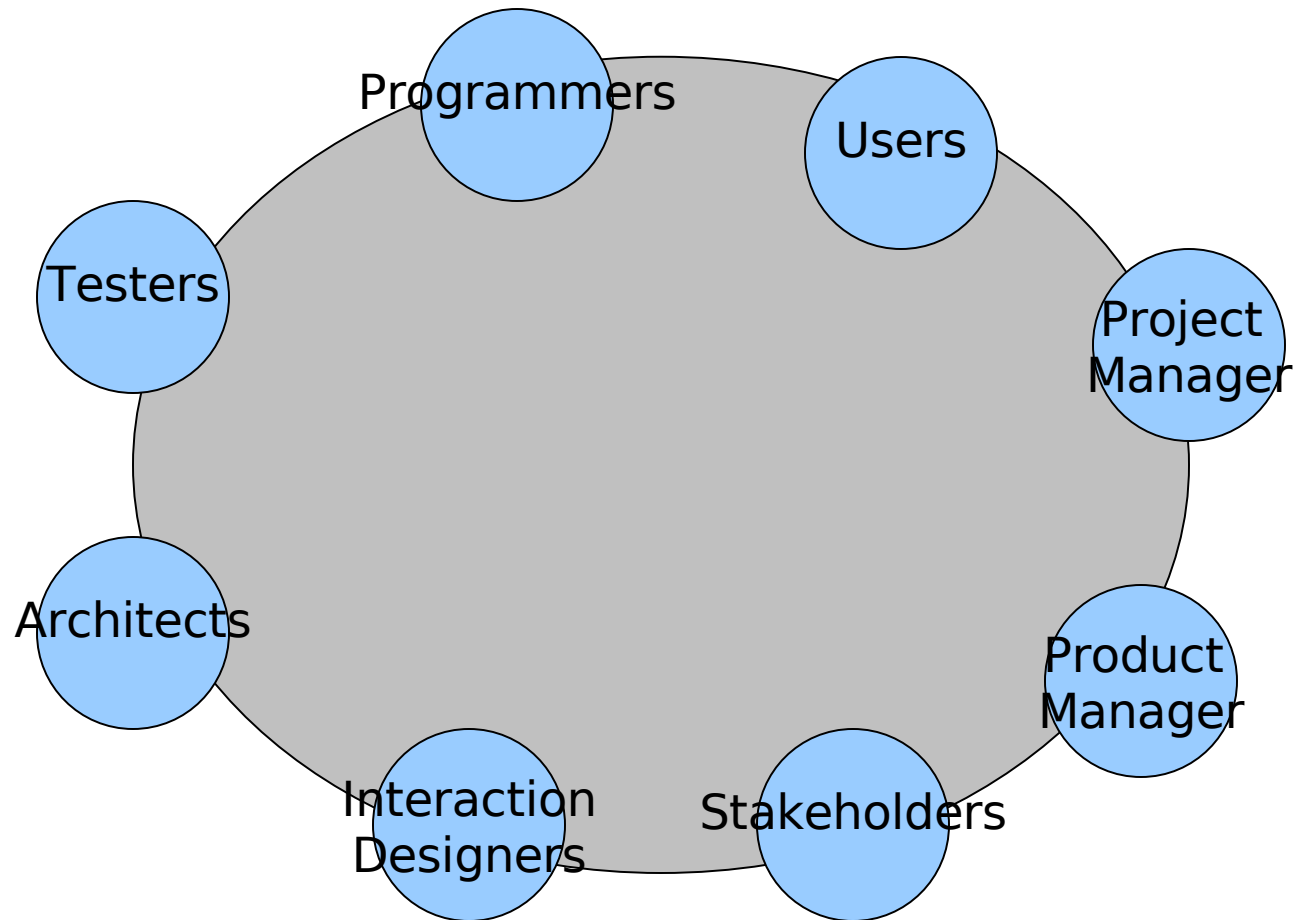


# The Whole XP Team

- Emphasis on the “Whole Team”
- Collaboration and collocation

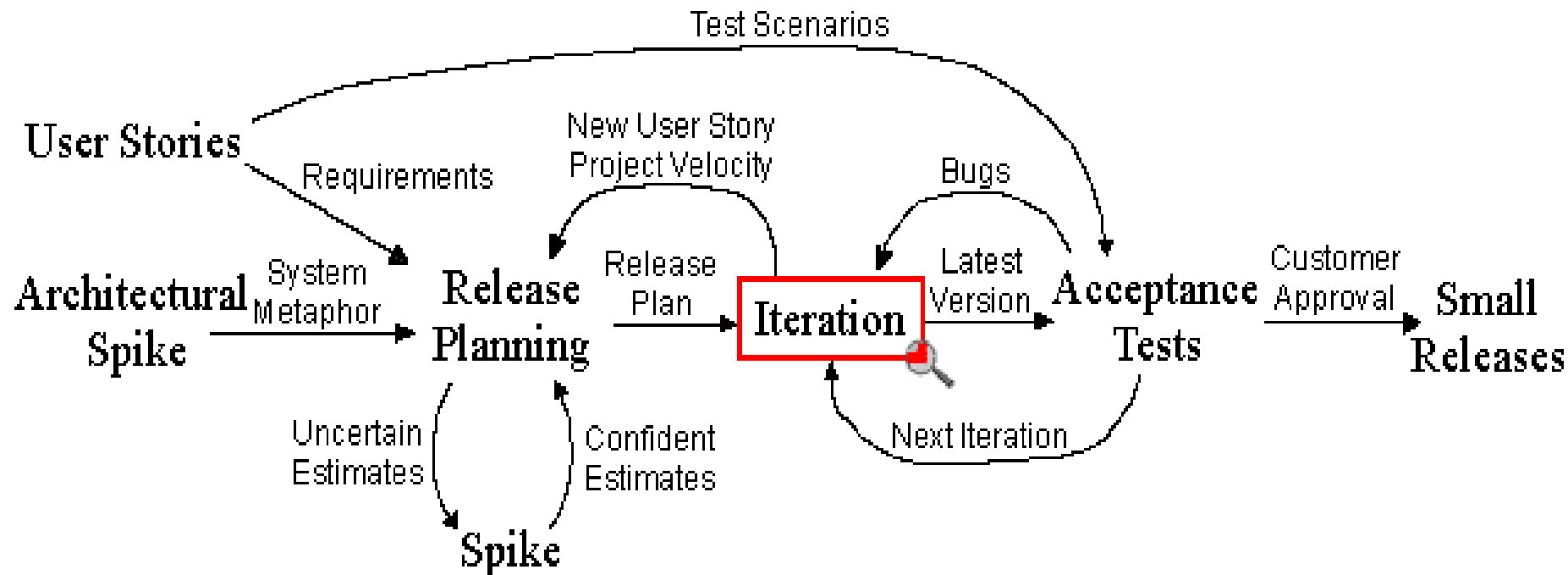


# The Whole XP Team



# XP Project

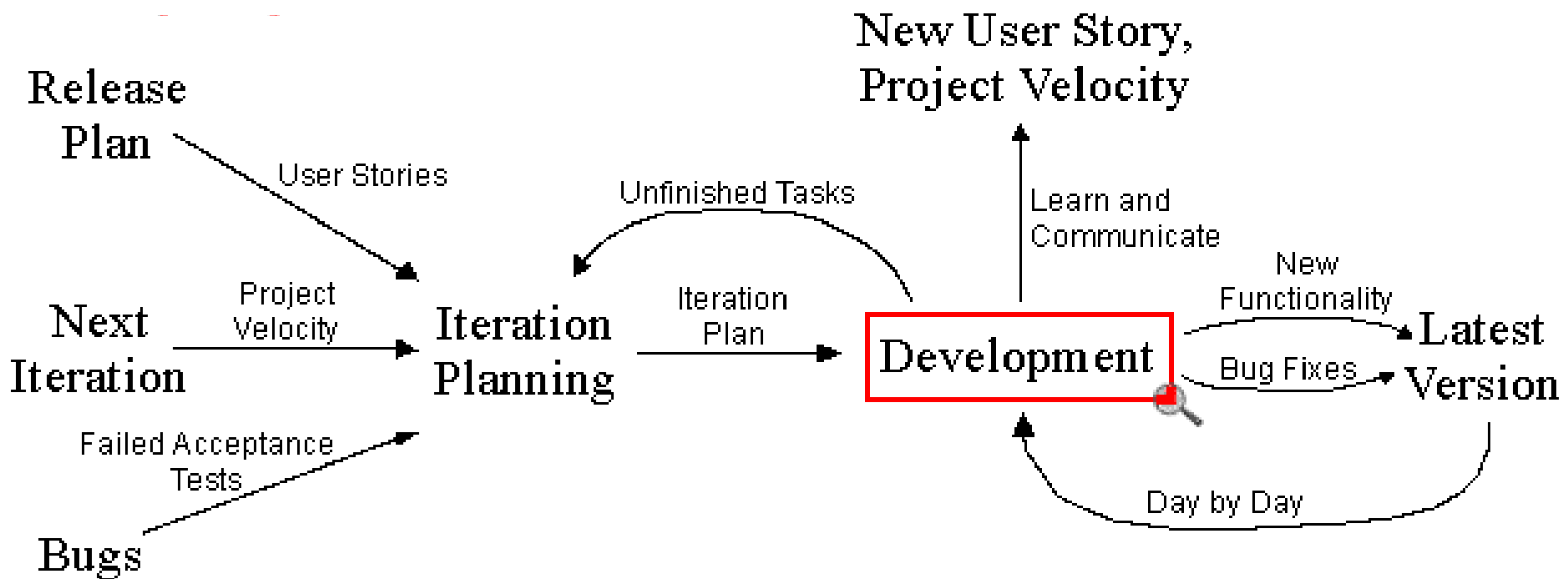
## Extreme Programming Project



# XP Project Iteration

## Iteration

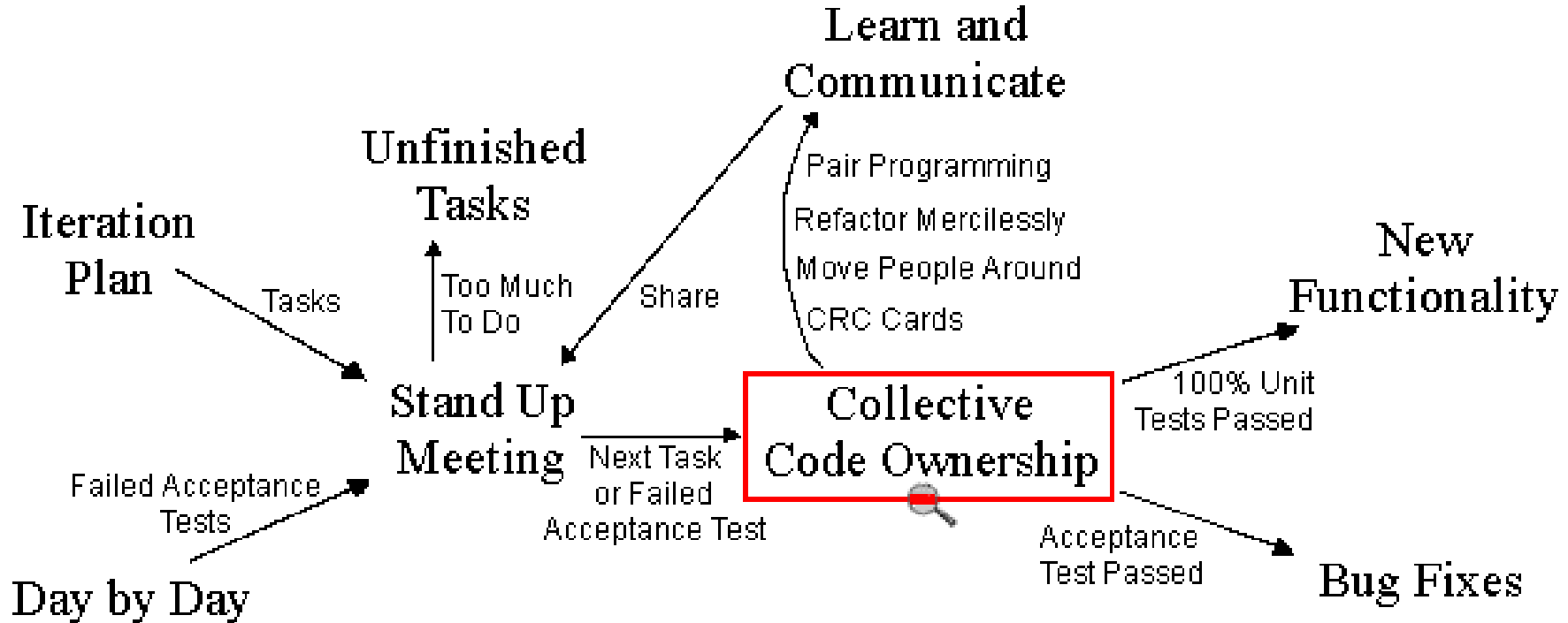
 Zoom Out



# XP Project Development

## Development

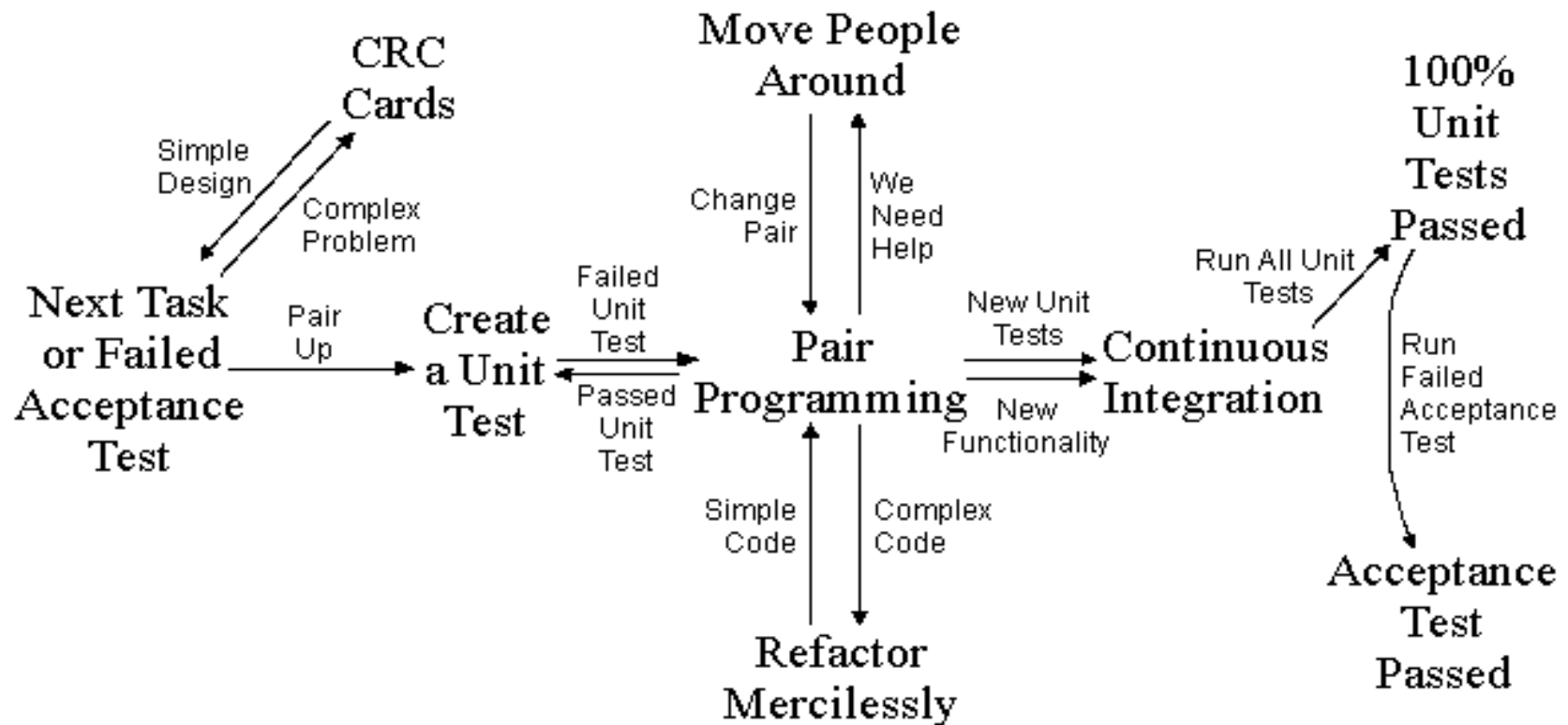
 Zoom Out



# XP Project Coding

## Collective Code Ownership

 Zoom Out



# FAQs

Some Frequently asked questions  
about XP

# XP vs. Rational Unified Process

	XP	RUP
Primary Communication Medium	Short Daily Face to Face meetings, Pair Programming	Client Status Meetings, Design Meetings
Development Methodology	Code, Refine, Test	Design, Document, Code, Test
Focuses on	Delivering software quickly	Following the process to develop good software
Quality Focus	Eliminate Defects as early as possible in the process using whole team	Eliminate defects on a scheduled basis with a separate team

# Scaling XP

- XP seems to allow smaller teams to accomplish an awful lot
- XP seems to hit single-team challenges around 12-16 developers
- XP can scale by building recursive teams
- Recommended to build small team first, incrementally grow, and use first team to seed recursive teams
- XP has been used on teams of 40-50



# Documentation

- XP is a minimalist process
- Many teams successful with index cards and large Post-Its®
- XP is not anti-documentation, but encourages doing the least amount that is really needed
- Document when needed for distributed sharing, historical needs, summarizing, etc.



# Adopting XP

- Find a suitable team and project
- Two strategies:
  - Adopt XP by-the-book first, then modify
  - Refactor your process incrementally to XP
- Be mindful of practice synergies
- Manage the team culture and change
- Allow sufficient time for feedback and learning
- An experienced coach can be valuable



# Questions

?