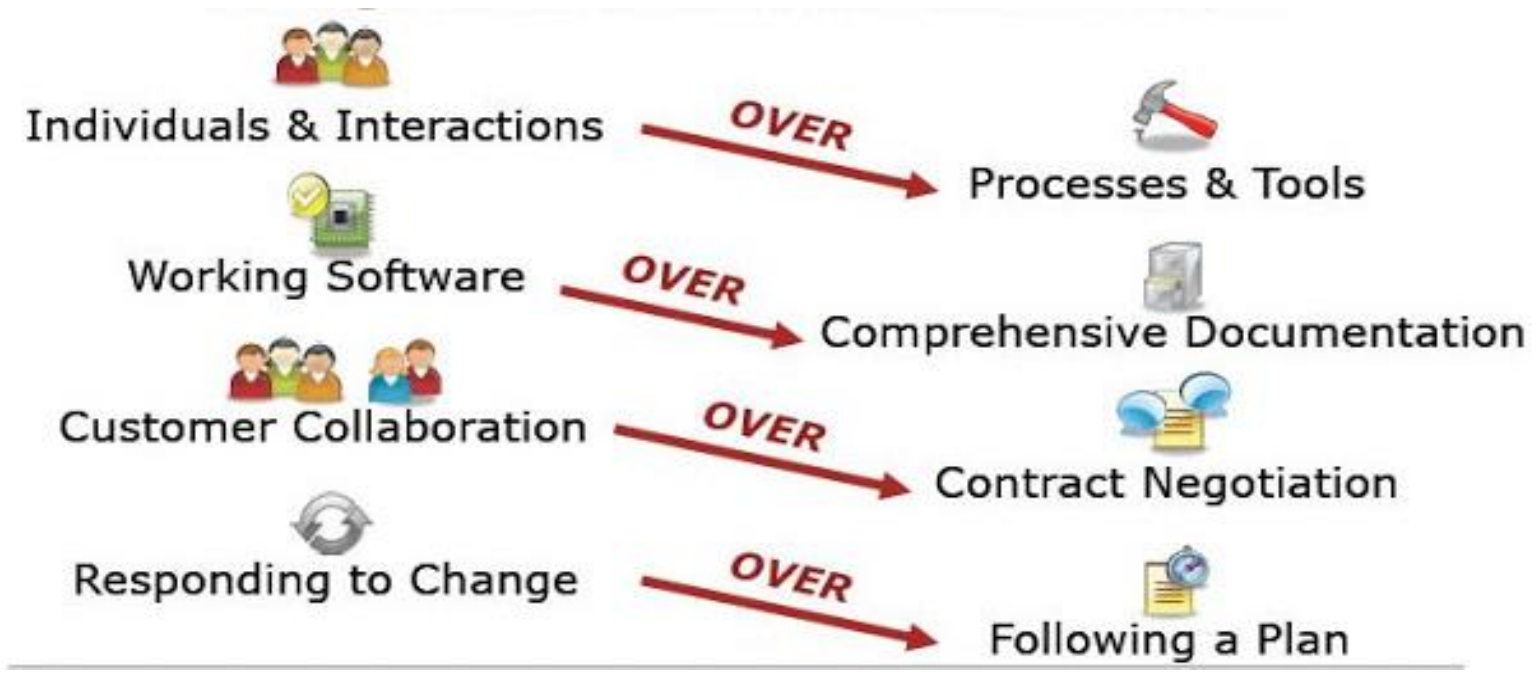
Agile development

eXtreme programming

What is Agile development?

An umbrella term covering development frameworks that adhere to the **Agile Manifesto**:

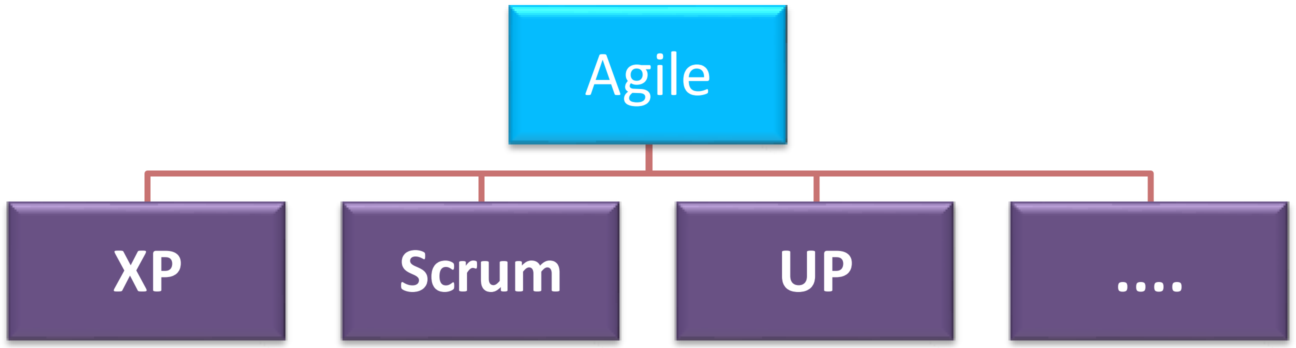
"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:



That is, while there is value in the items on the right, we value the items on the left more."

Agile Taxonomy

- Of course, given this loose definition of agile, many things fall into this category
- Some are more extreme in their definitions
- Hybrid approaches are now quite common, and can work well if they are sympathetic to their roots.

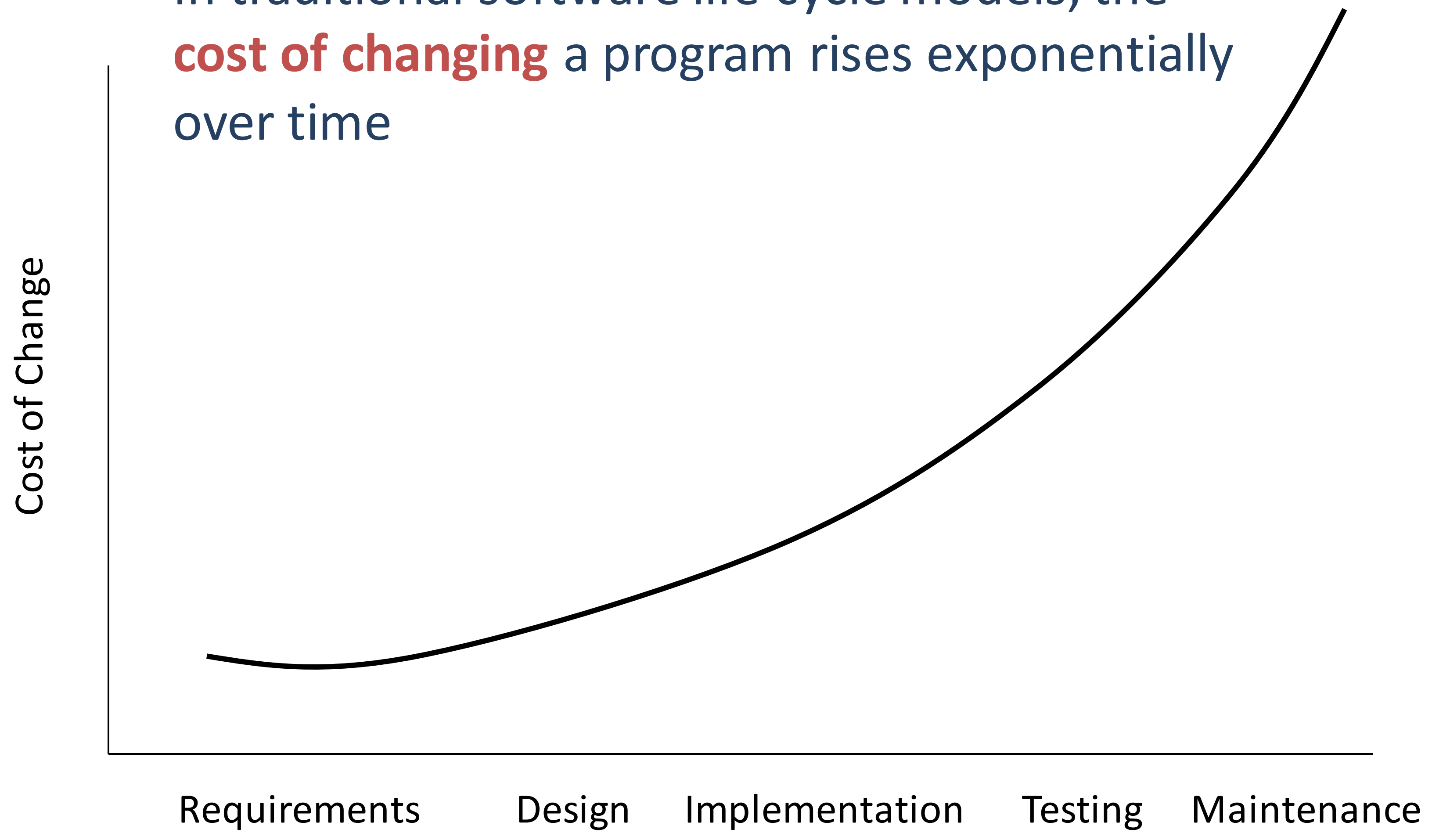


An Agile Development Methodology

- **XP = eXtreme Programming**
  - It does not encourage blind hacking. It is a systematic methodology.
- Developed by Kent Beck
  - XP is a light-weight methodology **for small to medium-sized teams** developing software in the face of vague or rapidly changing requirements.
- Alternative to "heavy-weight" software development models (which tend to avoid change and customers)
  - Rather than planning, analyzing, and designing for the future, XP programmers do all of these activities a little at a time throughout development.

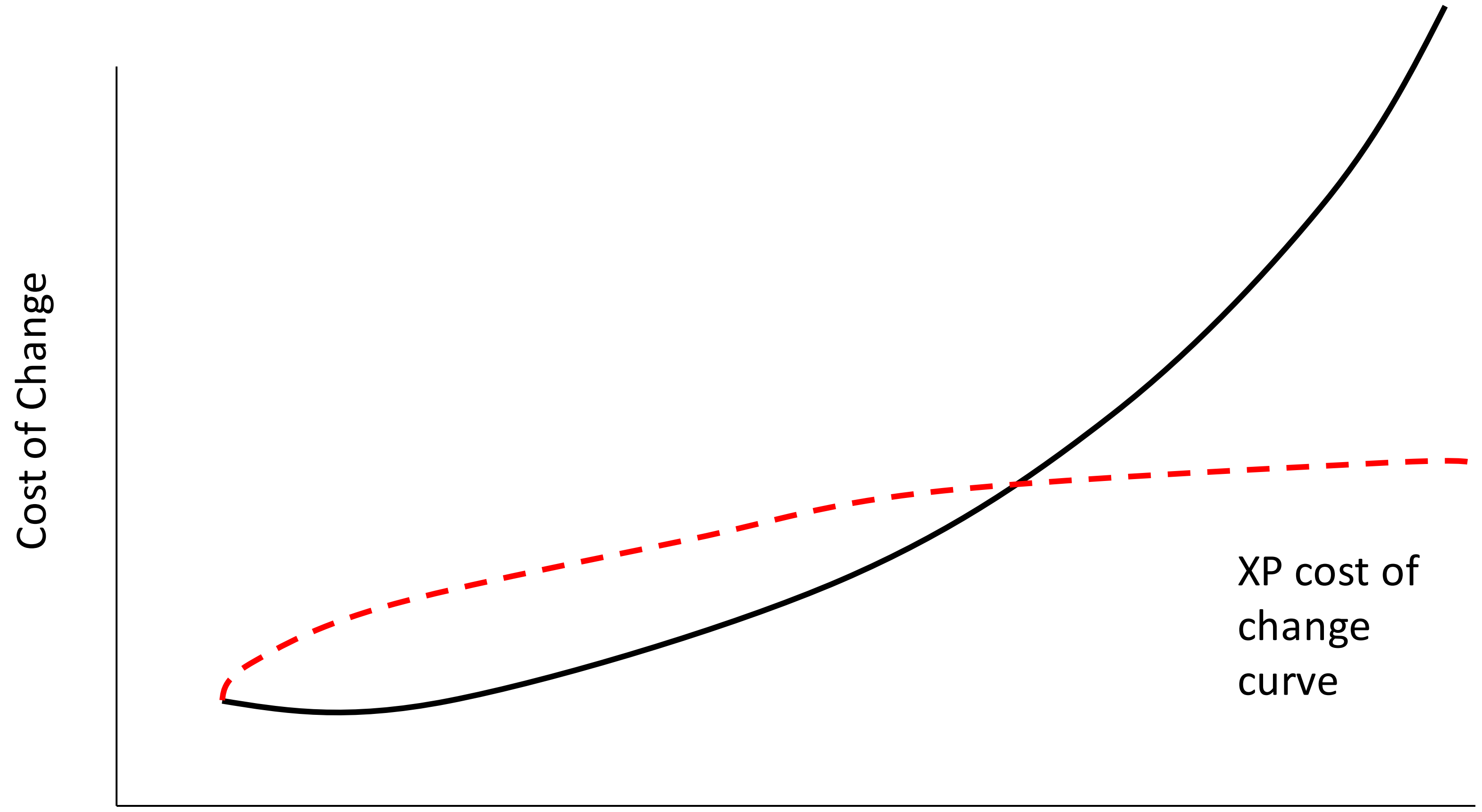
Traditional Processes are 'Heavy'

In traditional software life cycle models, the **cost of changing** a program rises exponentially over time

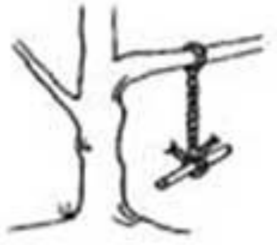


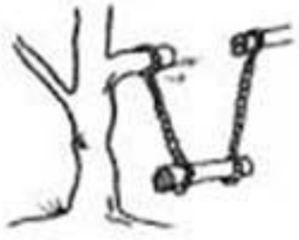
XP Cost of Change Curve

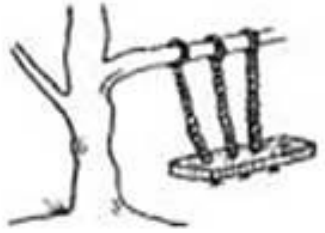
XP Practices flatten the cost of change curve.

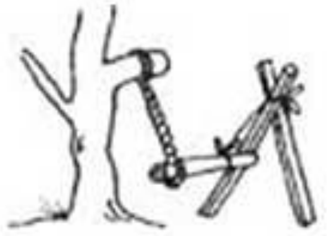


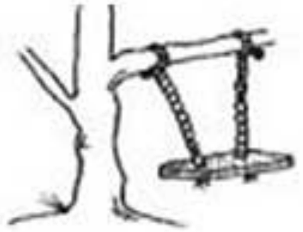
What can go wrong?

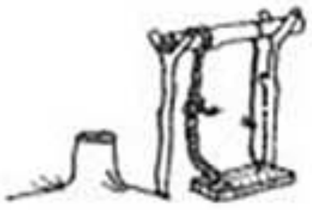

What the customer asked for


How the designer saw it


What the spec says


What the programmer wrote


What the customer really wanted


How it works

XP is a lightweight (agile) process:

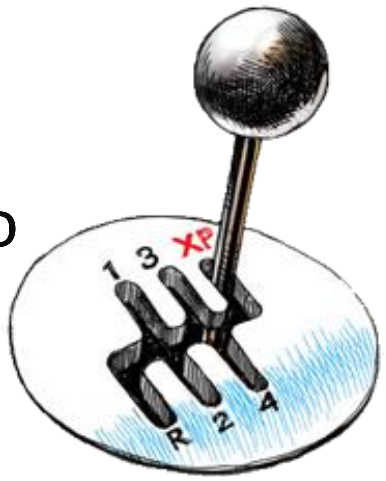
- Instead of lots of documentation nailing down what customer wants up front, XP emphasizes **plenty of feedback**
- **Embrace change**: iterate often, design and redesign, code and test frequently, keep the customer involved
- Deliver software to the customer in short (2 week) iterations
- Eliminate defects early, thus reducing costs

More on XP

- XP tends to use small teams, thus reducing communication costs.

- XP puts Customers and Programmers in one place.

- XP's practices work together in synergy, to get a team moving as quickly as possible to deliver value the customer wants



eXtreme…

Taking proven practices to the extreme….

- If testing is good, let everybody test all the time
- If code reviews are good, review all the time
- If design is good, refactor all the time
- If integration testing is good, integrate all the time
- If simplicity is good, do the simplest thing that could possibly work
- If short iterations are good, make them really, really short

PROCESS

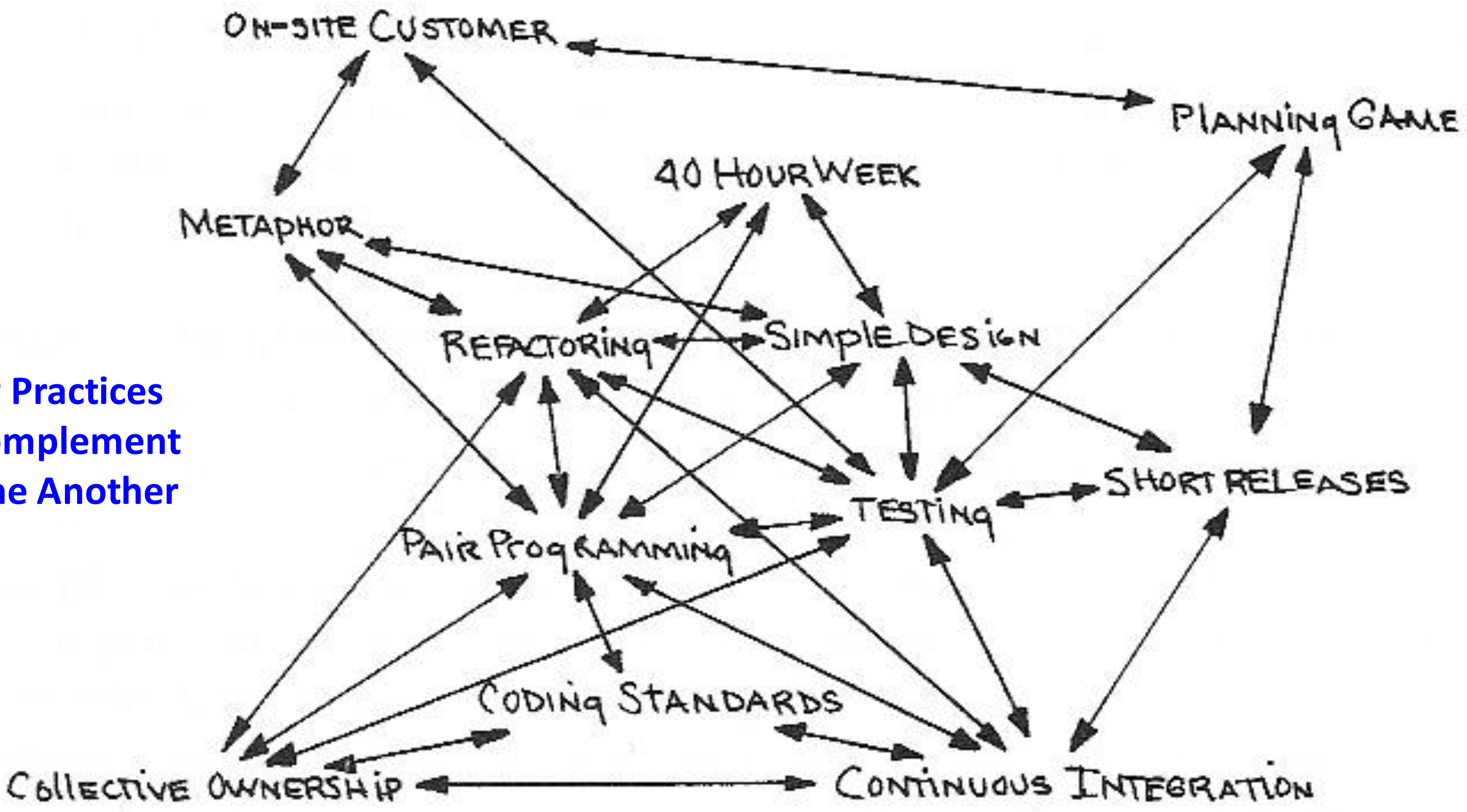
XP is an **iterative and incremental** process. The project is divided into smaller “mini-projects” which result in an increment of functionality, the so-called release.

A release is a version of the planned system that makes business sense. All features that are part of the release are implemented completely.

XP Practices

An XP team is expected to follow 12 simple practices:

XP Practices Complement One Another



Metaphor

- Extreme Programming teams develop a common vision of how the program works
- The metaphor is a simple description of how the program works, such as

  “this program works like a hive of bees, going out for pollen and bringing it back to the hive”

  as a description for an agent-based information retrieval system.

The Planning Game

- Customer comes up with a list of desired features for the system
- Each feature is written out as a **user story**
  - Describes in broad strokes what the feature requires
  - Typically written in 2-3 sentences on index cards
- Developers estimate how much effort each story will take, and how much effort the team can produce in a given time interval (iteration)

User Story Examples

A user wants access to the system, so they find a system administrator, who enters in the user's First Name, Last Name, Middle Initial, E-Mail Address, Username (unique), and Phone Number.

Risk: Low **Cost:** 2 points

The user must be able to search for a book.

Risk: High **Cost:** (too large!)

The user must be able to search for a book by Title, and display the results as a list.

Risk: Med. **Cost:** 1 point

The user must be able to search for a book by Category, and display the results as a list.

Risk: Med. **Cost:** 2 points

User stories template

User stories often follow a simple template:

***As a** <type of user>**, I want** <some goal> **so that** <some reason>*

Examples:

- **As a** user**, I want** to backup my entire hard drive, **so that** I can avoid loosing valuable data.

- **As a** user**, I want** to indicate folders not to backup, **so that** my backup drive isn't filled up with things I don't need saved.

- **As a** power user**, I want** to specify files or folders to backup based on file size, date created and date modified, **so that…..**

Estimation of User Stories

- **Project velocity** = how many hours can be committed to a project per week

  Volocity =idealworktime* othertasks factor

  Example : 20 = 40 * 0,5

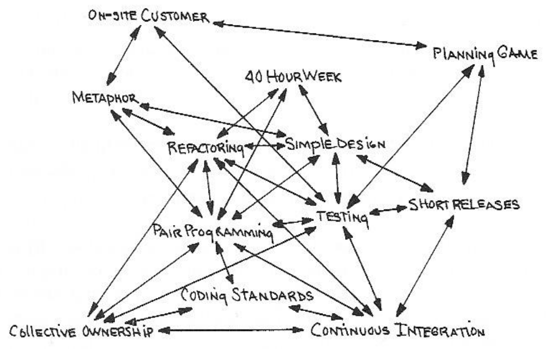
  The customer have 20 points to speed in an one week iteration

- Given developer estimates and project velocity, the customer prioritizes which stories to implement

User Stories

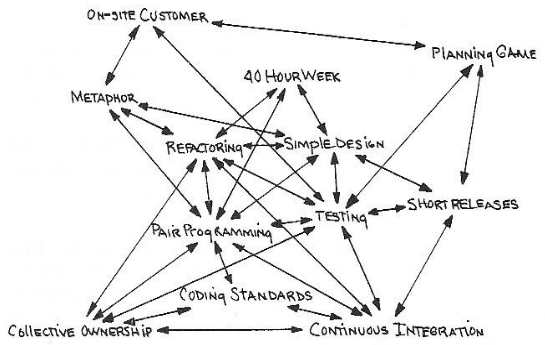
- **Drive the creation of the acceptance tests:**
  - There must be one or more tests to verify that a story has been properly implemented.
- **Different than Requirements:**
  - Should only provide enough detail to make a reasonably low risk estimate of how long the story will take to implement.
- **Different than Use Cases:**
  - Written by the Customer, not the Programmers, using the Customer's terminology

Small and simple

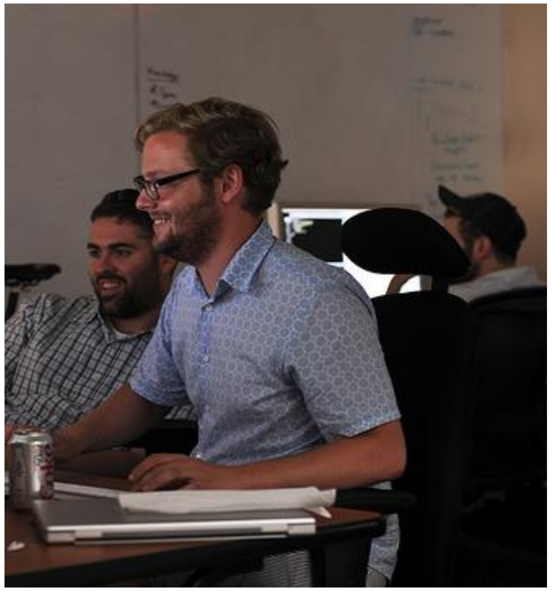


- **Small releases**
  - Start with the smallest useful feature set
  - Release early and often, adding a few features each time
- **Simple design**
  - Always use the simplest possible design that gets the job done.
    - What is the simplest thing that could possible work?
  - The requirements will change tomorrow, so only do what's needed to meet today's requirements

Pair programming



- Two programmers work together at one machine
- **Driver** enters code, while **navigator** critiques it
- Periodically switch roles



- **Research results:**
  - Pair programming increases productivity
  - Higher quality code (15% fewer defects) in about half the time (58%)

Exercise

Discuss with the student next to you:

- Can you think of other pair programming benefits?
- Can you think of any pair programming draw backs?

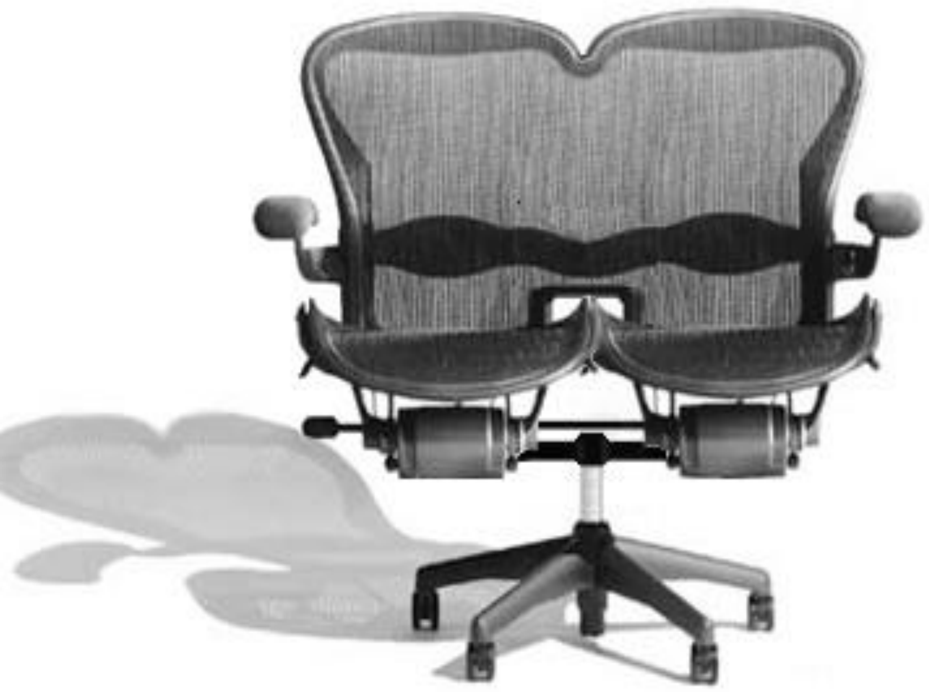
Pair Programming Benefits

- Continuous code review: better design, fewer defects
- Confidence to add to or change the system
- Discipline to always test and refactor
- Teach each other how the system works (reduced staffing risks)
- Learn from partner's knowledge and experience (enhances technical skills)

Pair Programming – Disadvantages

- Many tasks really don't require two programmers
- A hard sell to the customers
- Not for everyone

The most perfect chair for pair programming ☺



Key Features:

• Fully unit-tested in our ego-free ergonomics lab

•Essential office furniture for any eXtreme XP Pair (XXPP)

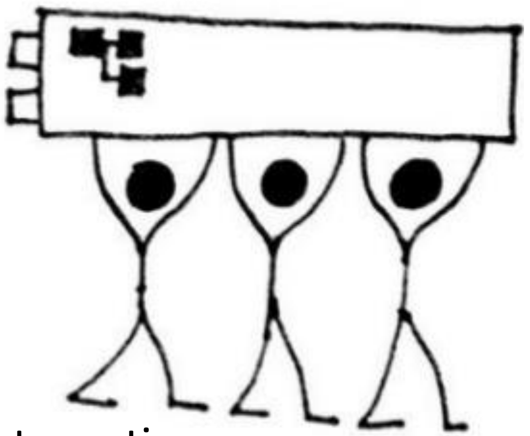
•Fully adjustable via individual or pair control

•Can be levered to standup-meeting height

•40-hour-week alarm buzzer built in

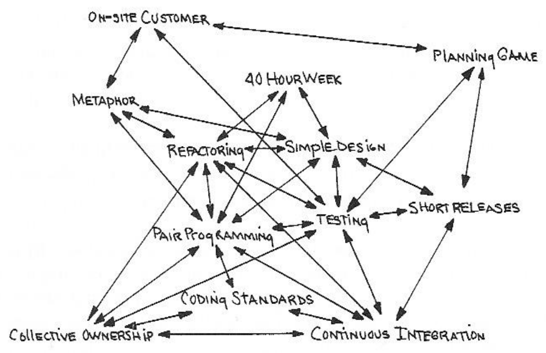
•Available in a range of attractive colors

More XP practices

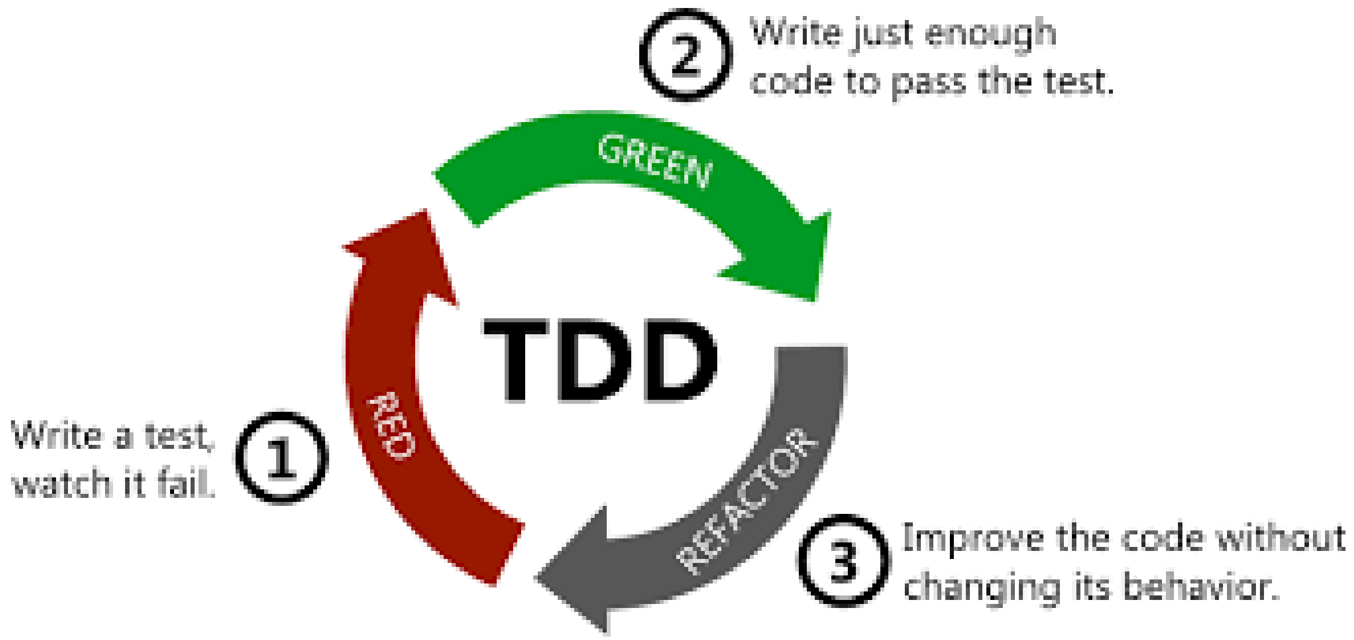


- Collective code ownership
  - No single person "owns" a module
  - Any developer can work on any part of the code base at any time
- Coding standards
  - Everyone codes to the same standards
  - Ideally, you shouldn't be able to tell by looking at it who on the team has touched a specific piece of code
- Continuous integration
  - All changes are integrated into the code base at least daily
  - Tests have to run 100% both before and after integration
  - Only one pair integrates code at a time.

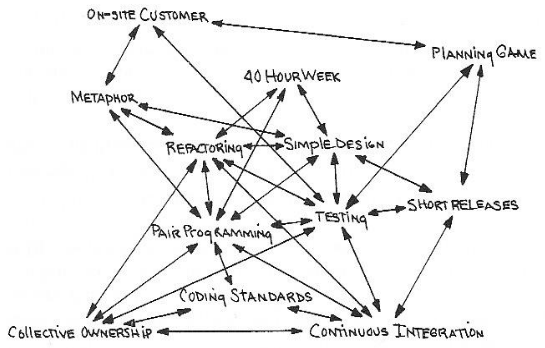
More XP practices



- Refactoring
  - Continuously improve quality of the code
  - You can do this with confidence that you didn't break anything because you have the tests


2 Write just enough code to pass the test.
GREEN
TDD
RED
Write a test, watch it fail. 1
REFACTOR
3 Improve the code without changing its behavior.

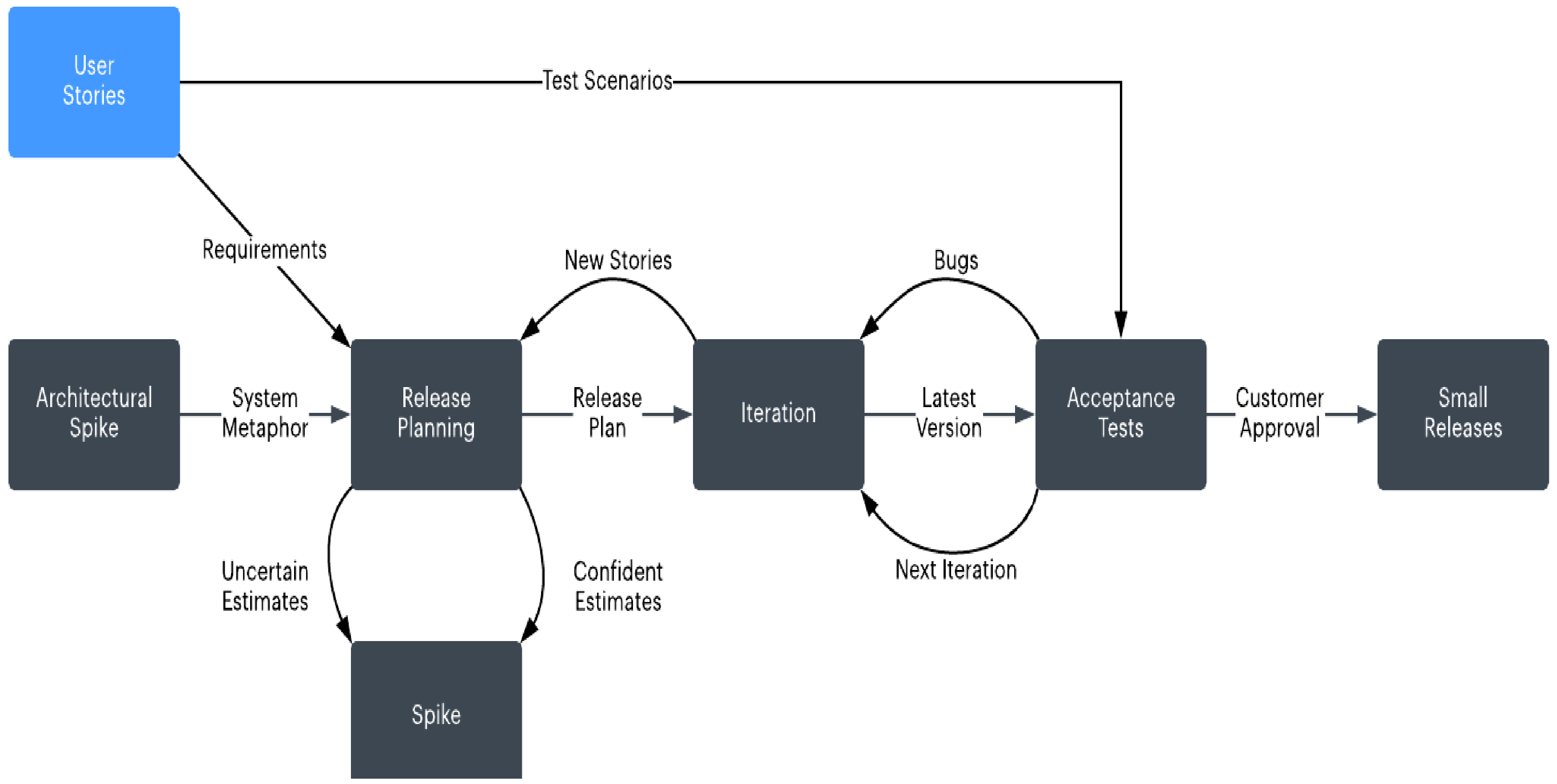
More practices



- 40-hour work week
  - Programmers go home on time
  - “fresh and eager every morning, and tired and satisfied every night”
  - up to one week of overtime is allowed
  - More than that and there’s something wrong with the process

- On-site customer
  - Development team has continuous access to a real live customer, that is, someone who will actually be using the system

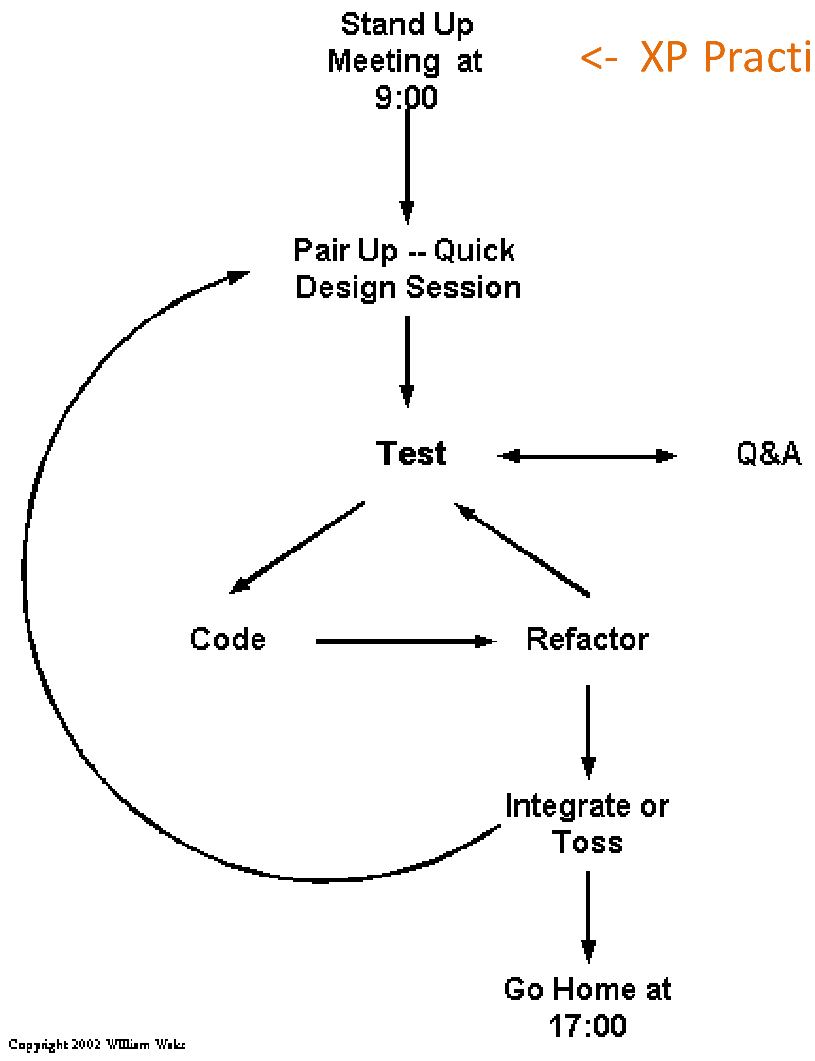
XP overview


User Stories
Test Scenarios
Requirements
New Stories
Bugs
Architectural Spike
System Metaphor
Release Planning
Release Plan
Iteration
Latest Version
Acceptance Tests
Customer Approval
Small Releases
Uncertain Estimates
Confident Estimates
Next Iteration
Spike

Spikes

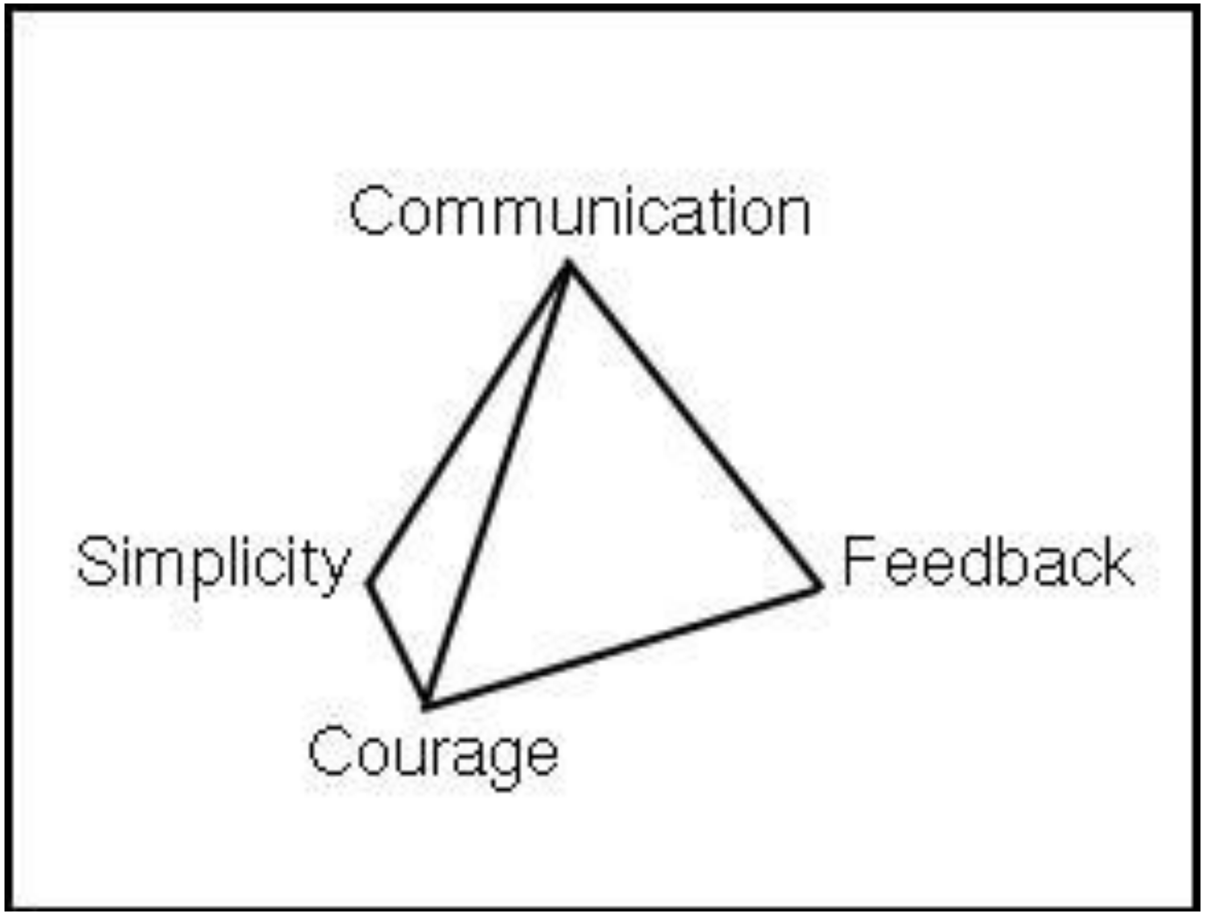
- Technology explorations
- Focus on high risk items
- Typically considered throw-away code
  - If not whole team must agree

The typical day of an XP Developer



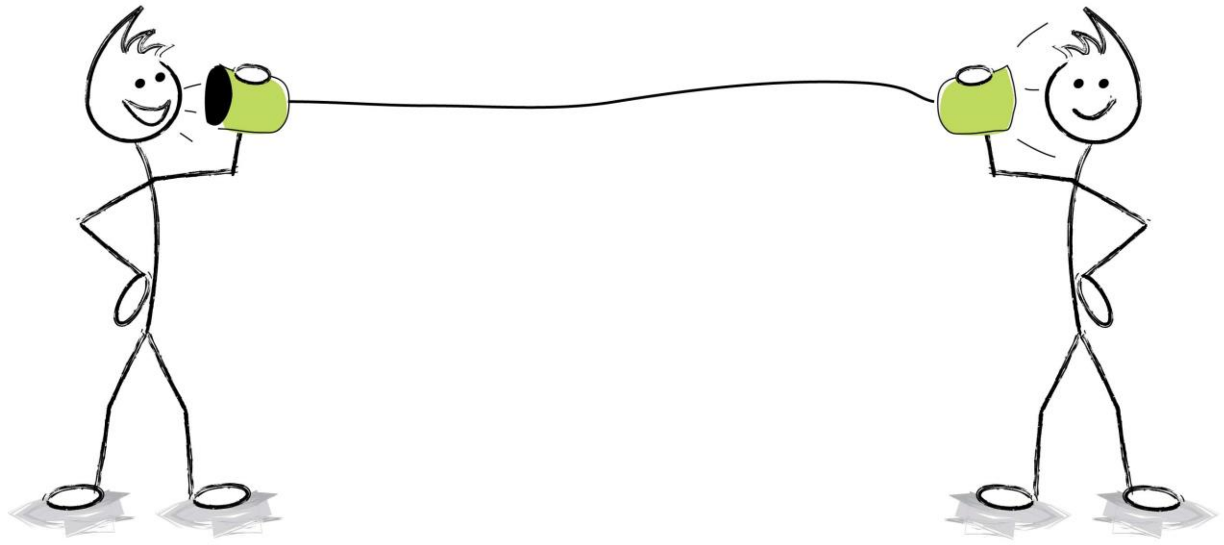
<- XP Practice #13

Four (5) Core Values of XP



The values reinforce each other to form a stable structure

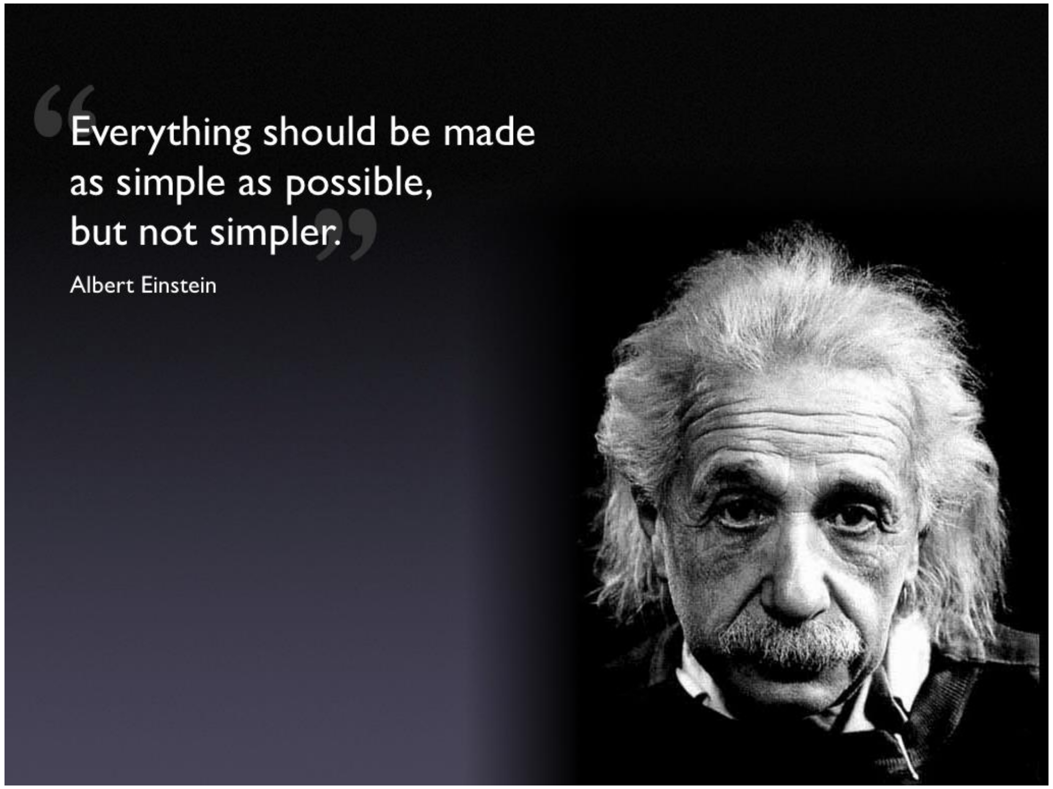
Communication



- XP emphasizes value of communication in many of its practices:
  - On-site customer, user stories, pair programming, collective ownership , daily standup meetings, etc.

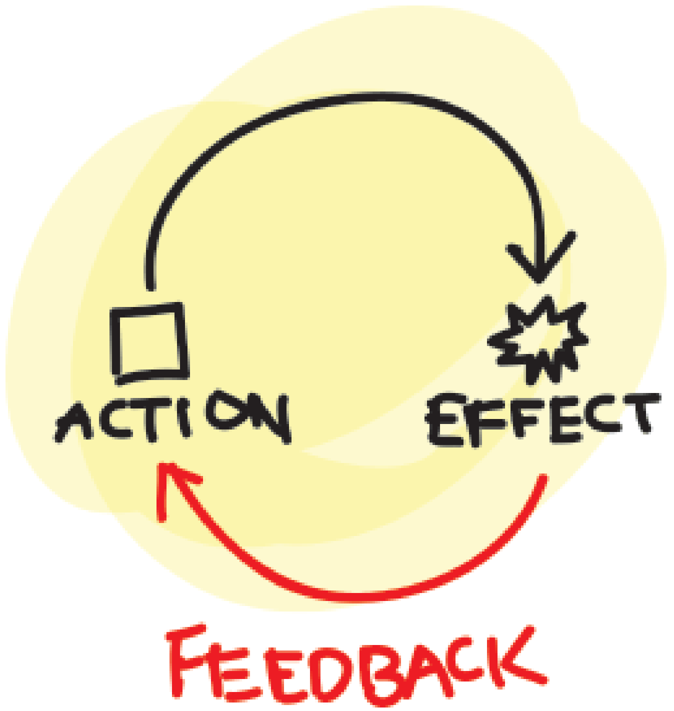
Simplicity

- "Do the simplest thing that could possibly work" principle
  - Elsewhere known as KISS


Everything should be made as simple as possible, but not simpler.
Albert Einstein

Feedback

- Unit tests tell programmers status of the system
- When customers write new *user stories,* programmers estimate time required to deliver changes
- Programmers produce new releases every 2-3 weeks for customers to review



Courage

- Courage to communicate and accept feedback
- Courage to throw code away
- Courage to refactor the architecture of a system

Who benefits from XP?

Programmers:

- get clear requirements & priorities
- can do a good job
- don’t work overtime

Customers:

- get most business value first
- get accurate feedback
- can make informed business decisions
- can change their mind

When to use XP

- Highly uncertain environments
- Internal projects
- Smaller projects

When not to use XP

- Large, complex environments
- Safety critical situations
- Well understood requirements
- Distant or unavailable customer

Key points

- Extreme programming includes practices such as systematic testing, continuous improvement and customer involvement.
- Customers are involved in developing requirements which are expressed as simple scenarios.
- The approach to testing in XP is a particular strength where executable tests are developed before the code is written.
- Key problems with XP include difficulties of getting representative customers and problems of architectural design.