INF 111 / CSE 121: Software Tools and Methods

Lecture Notes for Fall Quarter, 2007
Michele Rousseau
Set 6

(Some slides adapted from Susan E. Sim)

Announcements

- **Reminder: Quiz on Monday**
  - Lectures & Readings
- **Dropping?**
  - Friday Deadline for Dropping
    (the easy way)
- **Adding?**
  - I'll sign add cards on Friday provided others have dropped

Previous Lecture

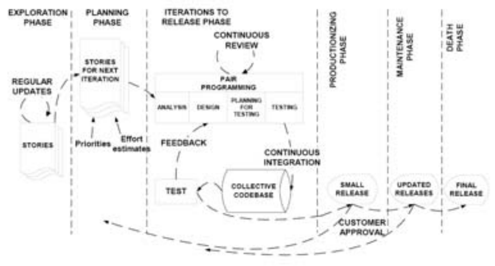
- **Finished the Agile Process Model**
- **Started on XP**

Today’s Lecture

- **Process Modeling**
  - Extreme Programming continues…
- **No Silver Bullet**

XP Lifecycle Model

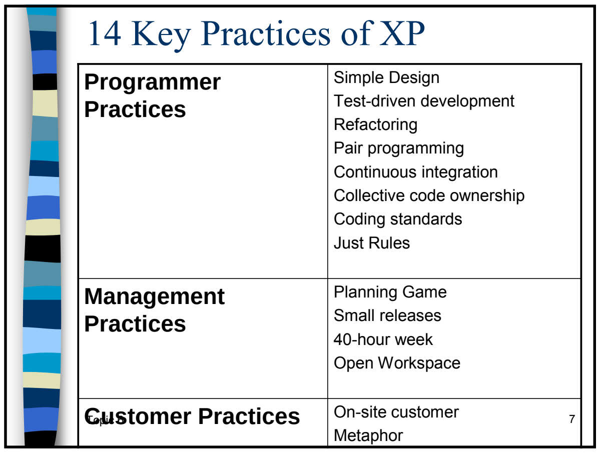
The life cycle of XP consists of five phases: Exploration, Planning, Iterations to Release, Productionizing, Maintenance and Death



Life cycle of the XP process.

5 Phases Of Development

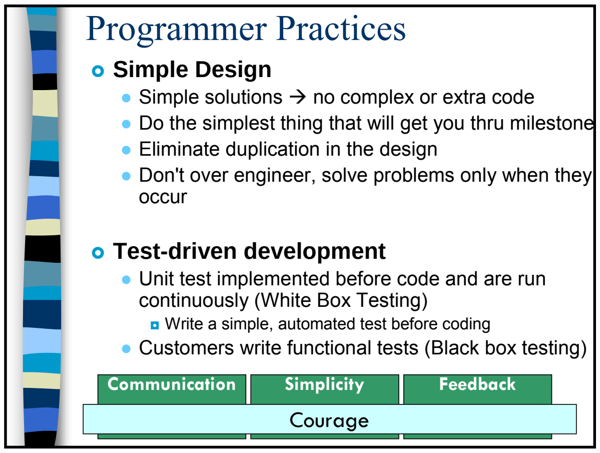
- **Exploration**
- **Planning**
- **Iterations to Release**
- **Productionizing**
- **Maintenance**
- **Death**



14 Key Practices of XP

Programmer Practices	Simple Design Test-driven development Refactoring Pair programming Continuous integration Collective code ownership Coding standards Just Rules
Management Practices	Planning Game Small releases 40-hour week Open Workspace
Customer Practices	On-site customer Metaphor

7

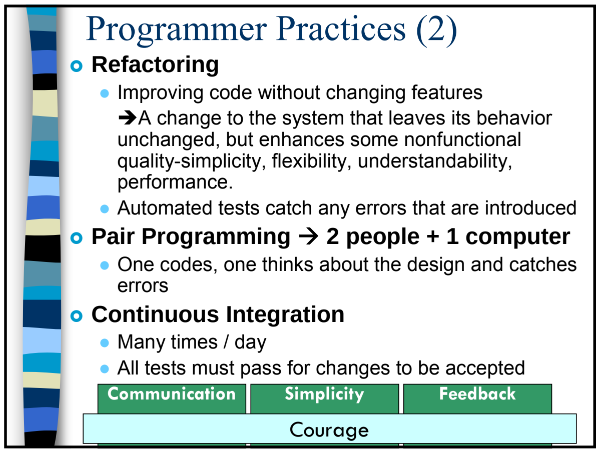


Programmer Practices

- **Simple Design**
  - Simple solutions → no complex or extra code
  - Do the simplest thing that will get you thru milestone
  - Eliminate duplication in the design
  - Don't over engineer, solve problems only when they occur
- **Test-driven development**
  - Unit test implemented before code and are run continuously (White Box Testing)
    - Write a simple, automated test before coding
  - Customers write functional tests (Black box testing)

Communication | Simplicity | Feedback

Courage



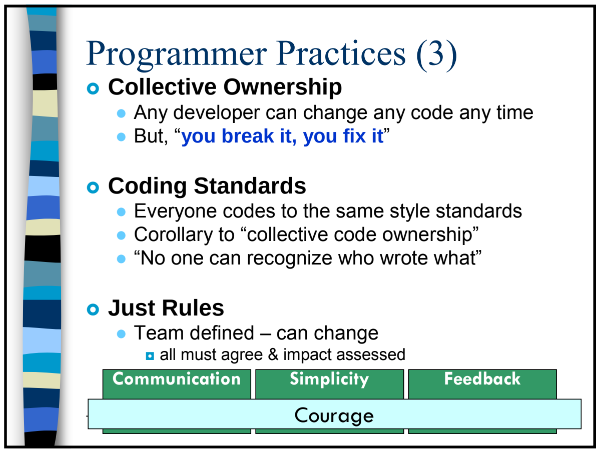
Programmer Practices (2)

- **Refactoring**
  - Improving code without changing features

    ➔A change to the system that leaves its behavior unchanged, but enhances some nonfunctional quality-simplicity, flexibility, understandability, performance.
  - Automated tests catch any errors that are introduced
- **Pair Programming → 2 people + 1 computer**
  - One codes, one thinks about the design and catches errors
- **Continuous Integration**
  - Many times / day
  - All tests must pass for changes to be accepted

Communication | Simplicity | Feedback

Courage

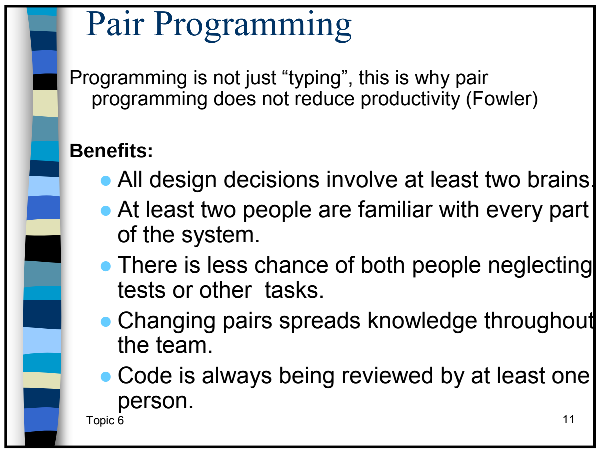


Programmer Practices (3)

- **Collective Ownership**
  - Any developer can change any code any time
  - But, “**you break it, you fix it**”
- **Coding Standards**
  - Everyone codes to the same style standards
  - Corollary to “collective code ownership”
  - “No one can recognize who wrote what”
- **Just Rules**
  - Team defined – can change
    - all must agree & impact assessed

Communication | Simplicity | Feedback

Courage



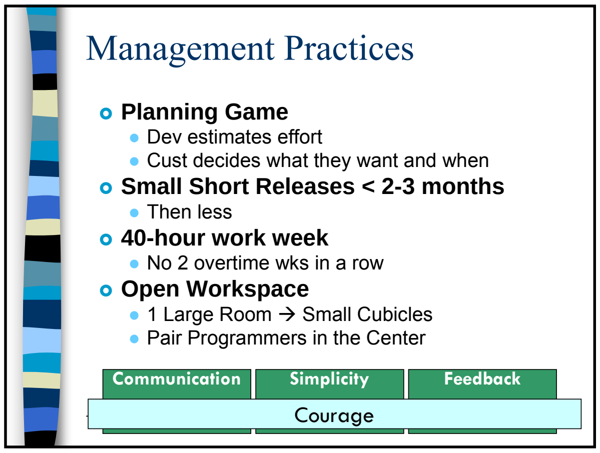
Pair Programming

Programming is not just “typing”, this is why pair programming does not reduce productivity (Fowler)

Benefits:

- All design decisions involve at least two brains.
- At least two people are familiar with every part of the system.
- There is less chance of both people neglecting tests or other tasks.
- Changing pairs spreads knowledge throughout the team.
- Code is always being reviewed by at least one person.

Topic 6 11



Management Practices

- **Planning Game**
  - Dev estimates effort
  - Cust decides what they want and when
- **Small Short Releases < 2-3 months**
  - Then less
- **40-hour work week**
  - No 2 overtime wks in a row
- **Open Workspace**
  - 1 Large Room → Small Cubicles
  - Pair Programmers in the Center

Communication | Simplicity | Feedback

Courage

Customer Practices

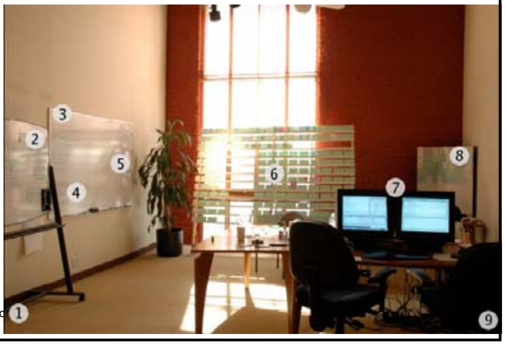
- **On-site customer**
  - Need customer/user around to answer questions
  - Builds a bond, working relationship
- **Metaphors**
  - "Shared Story" guides development
  - Describes how system should work

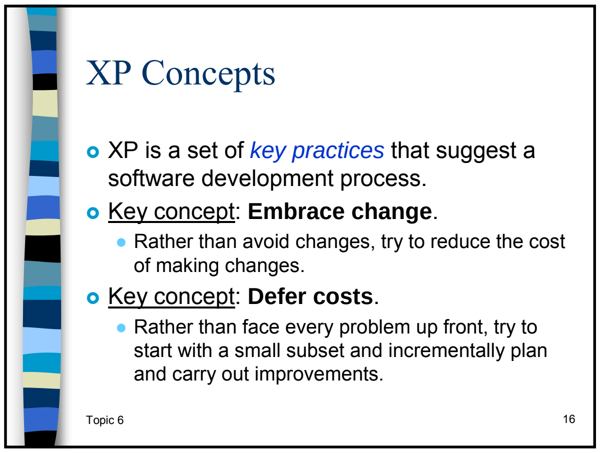
Communication	Simplicity	Feedback
Courage		

User Story / User Card

http://www.scissor.com/resources/teamroom/

The XP Team Room

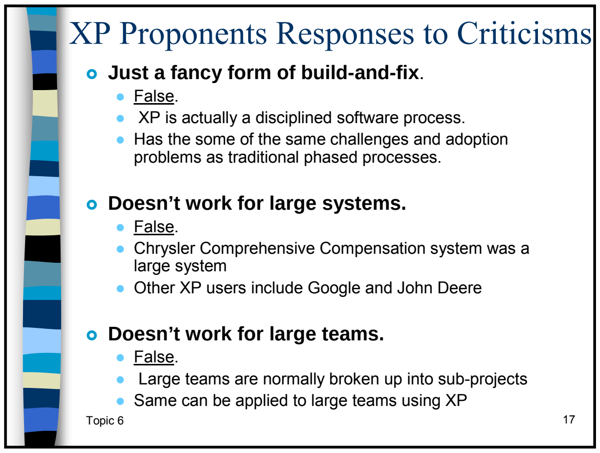




XP Concepts

- XP is a set of *key practices* that suggest a software development process.
- Key concept: **Embrace change**.
  - Rather than avoid changes, try to reduce the cost of making changes.
- Key concept: **Defer costs**.
  - Rather than face every problem up front, try to start with a small subset and incrementally plan and carry out improvements.

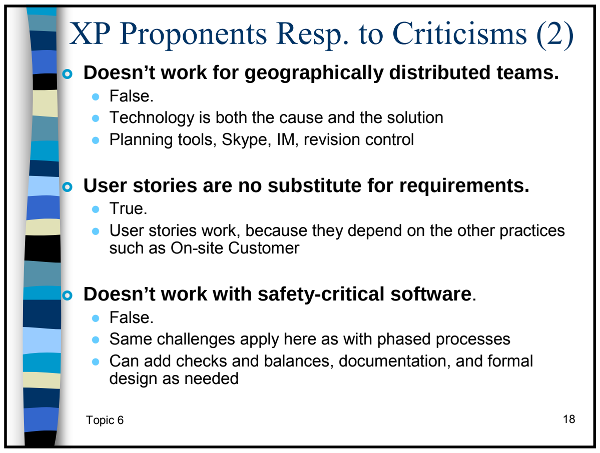
Topic 6 16



XP Proponents Responses to Criticisms

- **Just a fancy form of build-and-fix**.
  - False.
  - XP is actually a disciplined software process.
  - Has the some of the same challenges and adoption problems as traditional phased processes.
- **Doesn't work for large systems.**
  - False.
  - Chrysler Comprehensive Compensation system was a large system
  - Other XP users include Google and John Deere
- **Doesn't work for large teams.**
  - False.
  - Large teams are normally broken up into sub-projects
  - Same can be applied to large teams using XP

Topic 6 17



XP Proponents Resp. to Criticisms (2)

- **Doesn't work for geographically distributed teams.**
  - False.
  - Technology is both the cause and the solution
  - Planning tools, Skype, IM, revision control
- **User stories are no substitute for requirements.**
  - True.
  - User stories work, because they depend on the other practices such as On-site Customer
- **Doesn't work with safety-critical software**.
  - False.
  - Same challenges apply here as with phased processes
  - Can add checks and balances, documentation, and formal design as needed

Topic 6 18

XP Proponents Resp. to Criticisms (3)

- **Doesn't produce documentation.**
  - Maybe. XP only produces as much documentation as is needed, when it is needed (simplicity).
- **It is wasteful, because you're doing constantly doing re-design.**
  - False.
  - Planning everything up front is wasteful, because things are going to change anyways.
- **Not suitable for all projects**
  - True.
  - User functionality is simple, algorithms hard
  - Example: scientific applications

Topic 6 19

Productivity Gains

- **For a Web Dev Project**
  - 66% increase in new lines of code produced
  - 302% inc in new methods developed
  - 283% inc in # of new classes implemented

Topic 6 **Maruer & Martel 2002b** 20

Cons

- **Corp Culture must support XP**
  - Any resistance can lead to failure
- **Best for teams < 20**
- **Best if teams are collocated**
  - On the same floor
- **Technology that does not support "graceful change" → may not be suitable**

Topic 6 21

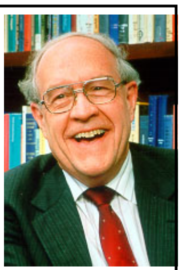
More Reading if you are interested

- **Agile**
  - Abrahamsson, P, et al. (2002). Agile software development methods: Review and analysis. VTT Publications 478.
  - http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf
- **XP**
  - Beck, K. (1999). Extreme programming explained: Embrace change. Reading Mass., Addison-Wesley

The Mythical Man-Month

- **Originally Published in 1975**
  - Fred Brooks
  - Based on Experiences From OS/360 in mid-60's
- **So why should we care?**
- **Some interesting Stats**
  - Amazon.com Sales Rank:
    #3,201 in Books
    #1 in Microprocessor Design
    #3 in Systems Analysis & Design
    #12 in Software Engineering

Who is Fred Brooks?



- "Father of IBM OS/360"
- 1992 Computer Pioneer Award (IEEE)
- 1999 Turing award winner
- 2007 Harvard Centennial Medal
- Founded UNC-Chapel Hill CS dept

No-Silver Bullet

"There is *no single development,* in either technology or management technique, which by itself *promises even one order-of-magnitude improvement within a decade* in productivity, in reliability, in simplicity"

Topic 6 25

Essence & Accident

- **Essential Tasks**
  - Specifications, design & testing of conceptual constructs
- **Accidental (or incidental) Tasks**
  - Programming & Compiling

The essential tasks are the hard part.

Topic 6 26

Why is building s/w difficult?

"I believe that hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation"

- **It is the nature of s/w – inherent in the process**
- **Conceptual errors are the problem**

Topic 6 27

Four Inherent Difficulties

- **Complexity**
- **Conformity**
- **Changeability**
- **Invisibility**

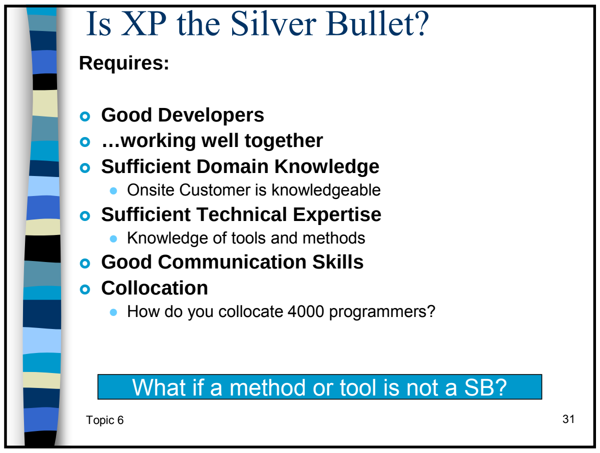
Complexity

- **Very large # of states**
- **Scaling is up is not a repetition of the same elements in large sizes**
- **Elements interact in a non-linear fashion**
- **Complexity → Communication**
- **It is difficult to extend large programs without creating side effects**

Complexity makes management difficult
Personnel turnover can be a disaster

Some of Brooks Suggestions

- **IF an OTS fits – buy it**
  - Why re-invent the wheel
- **Requirements refinement and rapid prototyping**
  - Many iterations between client and designer
- **Grow – don't build – software**
  - Develop incrementally
- **Train great designers**



Is XP the Silver Bullet?

Requires:

- **Good Developers**
- **…working well together**
- **Sufficient Domain Knowledge**
  - Onsite Customer is knowledgeable
- **Sufficient Technical Expertise**
  - Knowledge of tools and methods
- **Good Communication Skills**
- **Collocation**
  - How do you collocate 4000 programmers?

What if a method or tool is not a SB?

Topic 6 31