INF 111 / CSE 121: Software Tools and Methods

Lecture Notes for Fall Quarter, 2007
Michele Rousseau
Set 5

(Some slides adapted from Susan E. Sim)

Announcements

- **Lab Ending Times**
- **Add/Drop revisited**
- **Quiz on Friday?**
- **First Assignment**

Previous Lecture

- **Finished Up Methods & Tools**
- **Started the Agile Process Model**

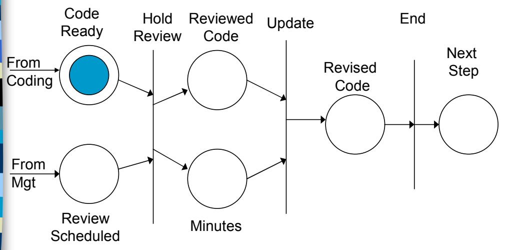
Today’s Lecture

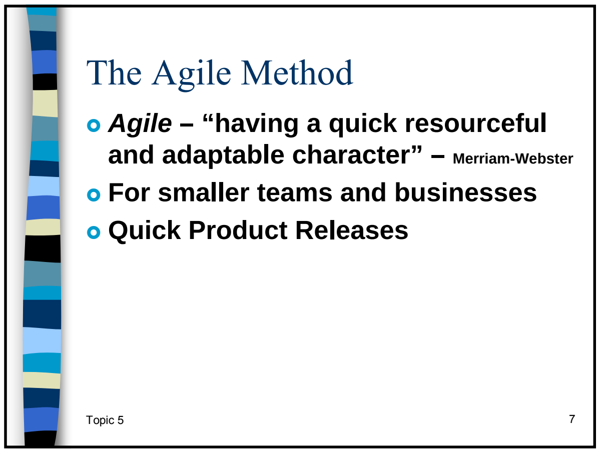
- **Process Modeling**
  - Agile Process Model
  - Extreme Programming

Process-Centered S/E Environment (PSEE)

- **Supports the entire Development Process**
- **Closely Tied to Process Modeling**
  - Petri-Nets
  - State Transition Diagrams
  - Etc…
- **Tends to support Back-End (Imp. & Testing)**
  - Easier to Formalize

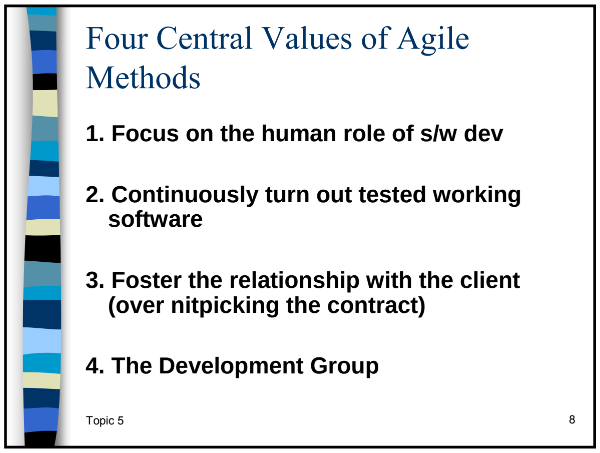
Petri-Net View of PSEE





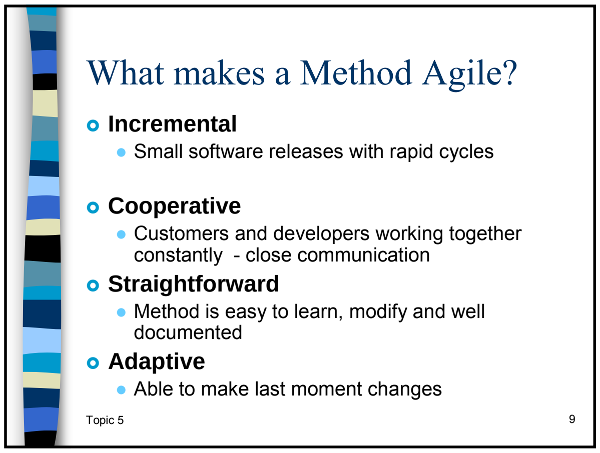
The Agile Method

- *Agile* – “having a quick resourceful and adaptable character” – Merriam-Webster
- For smaller teams and businesses
- Quick Product Releases



Four Central Values of Agile Methods

1. Focus on the human role of s/w dev
2. Continuously turn out tested working software
3. Foster the relationship with the client (over nitpicking the contract)
4. The Development Group



What makes a Method Agile?

- Incremental
  - Small software releases with rapid cycles
- Cooperative
  - Customers and developers working together constantly - close communication
- Straightforward
  - Method is easy to learn, modify and well documented
- Adaptive
  - Able to make last moment changes

How is Agile Different

- **"What is new about agile methods is not the practices they use but their recognition of people as the primary drivers of project success, coupled with an intense focus on effectiveness and maneuverability. This yields a new combination of values and principles that define an *agile* world view"**

Highsmith anc Cockburn (2001, p 122)

Agile vs. Traditional Plan-driven

Home-Ground Area	Agile Methods	Plan-driven Methods
Developers	Agile, knowledgeable, collocated, & collaborative	Plan-Oriented, adequate skills, access to external knowledge
Customers	Dedicated, knowledgeable, *collocated*, collaborative, representative, & empowered	Access to knowledgeable, collaborative, representative, and empowered customers

Agile vs. Traditional Plan-driven

Home-Ground Area	Agile Methods	Plan-driven Methods
Requirements	Largely emergent; rapid change	Knowable early; largely stable
Architecture	Designed for current requirements	Designed for current and *foreseeable* requirements
Refactoring	Inexpensive	Expensive
Size	Smaller teams and Products	Larger Teams and Products
Primary Objective	Rapid Value	High Assurance

Examples of Agile Methods

- **XP → Extreme Programming**
- **Scrum →**
  - ”Getting out-of play ball back into the game”
- **FDD → Feature Driven Development**
- **RUP → Rational Unified Process**

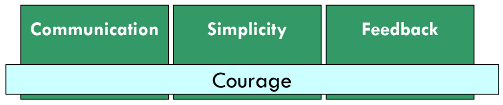
Extreme Programming (XP)

- **Invented by Kent Beck in 1996**
  - “Seat of the pants” fix to Chrysler project
  - To fix problems caused by long development cycles of traditional process models
- **Beck Published in 1999**

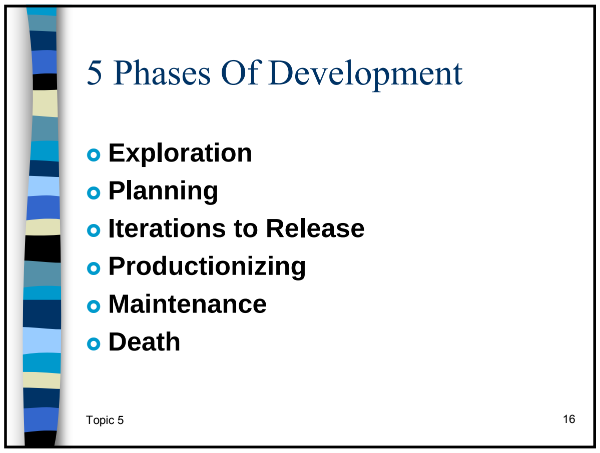
  “Extreme Programming Explained: Embrace Change”
  - Current hot topic in S/W Process
  - Loved and Hated
  - Tries to associate s/w process with eXtreme sports
- **Idea: Take a good programming practice and push it to the extreme**
  - Eg. Testing
  - Testing is good so… do it all the time

Premise of XP

- **The Four Values**



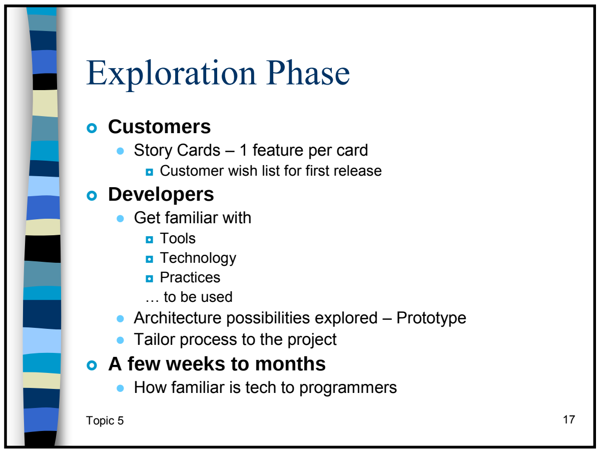
Hmmm.. But aren’t these standard “Best Practices”?
What’s new here?



5 Phases Of Development

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

Topic 5 16

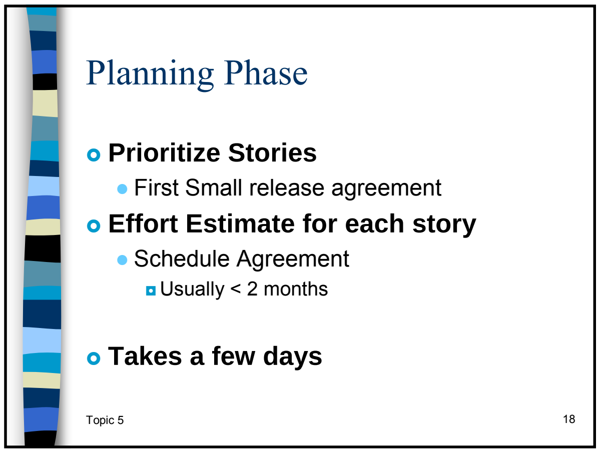


Exploration Phase

- **Customers**
  - Story Cards – 1 feature per card
    - Customer wish list for first release
- **Developers**
  - Get familiar with
    - Tools
    - Technology
    - Practices

    … to be used
  - Architecture possibilities explored – Prototype
  - Tailor process to the project
- **A few weeks to months**
  - How familiar is tech to programmers

Topic 5 17



Planning Phase

- **Prioritize Stories**
  - First Small release agreement
- **Effort Estimate for each story**
  - Schedule Agreement
    - Usually < 2 months
- **Takes a few days**

Topic 5 18

Iterations to Release Phase

- **Several Iterations before 1st Release**
- **# of Iterations determined in planning phase**
- **Each iteration takes 1-4 wks to implement**
- **Select stories wisely**
  - these enforce system architecture for the entire system
  - Customer chooses stories for each iteration
- **Functional tests created by Customer**
  - Run at the end of each iteration

At the end of last iteration → Production

Topic 5 19

Productionizing Phase

- **End testing before release**
- **New changes may be found**
  - Decide whether to include in current release
  - Documented for later implementation
    → Maintenance Phase
- **Iterations shortened**

Topic 5 20

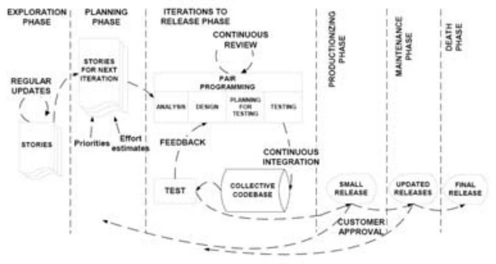
Maintenance and Death Phases

- **Maintenance**
  - May need more people
    - Maintain current production
    - Produce new Iterations
    - Change team structure
  - Development slows
- **Death Phase**
  Either…
  - All stories complete & quality is satisfactory
  - Not delivering expected outcomes
  - Too expensive to continue

Topic 5 21

XP Lifecycle Model

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



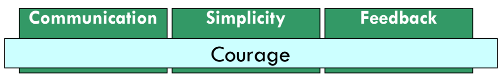
Life cycle of the XP process.

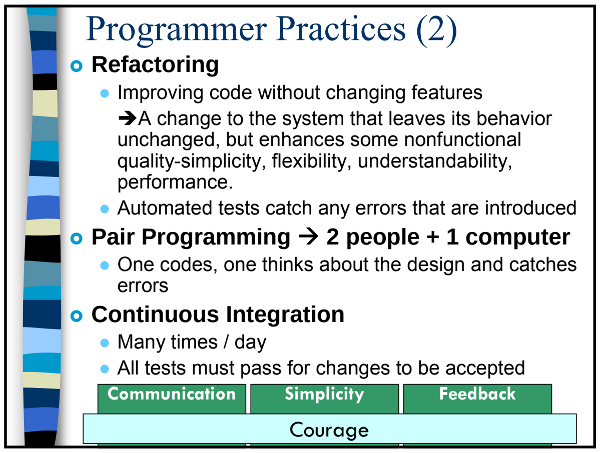
14 Key Practices of XP

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

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)





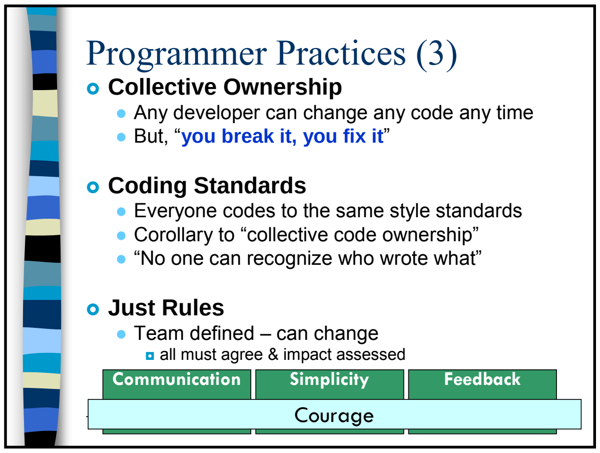
Programmer Practices (2)

- **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

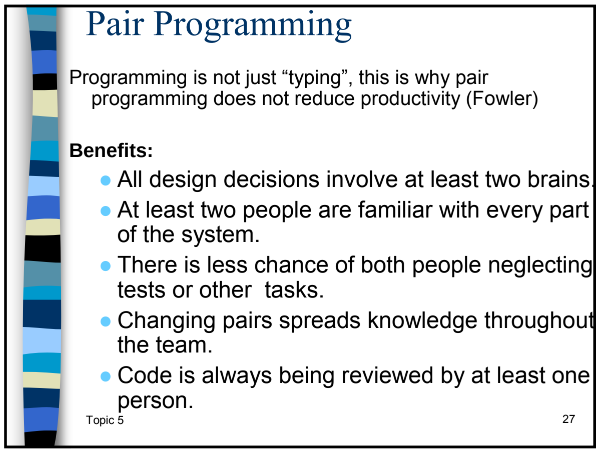


Programmer Practices (3)

- **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 5 27

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

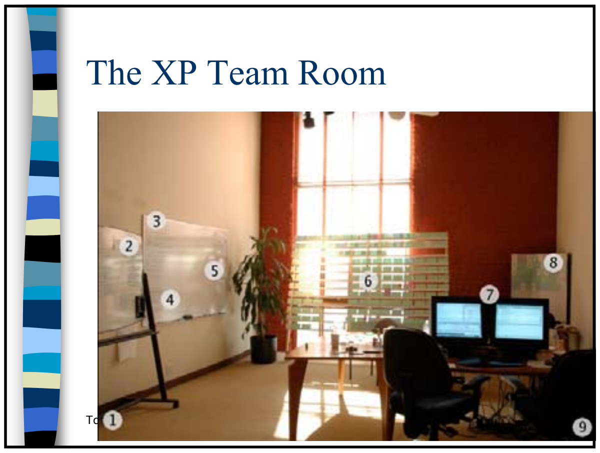
- **On-site customer**
  - Need customer/user around to answer questions
  - Builds a bond, working relationship
- **Metaphors**
  - “Shared Story” guides development
  - Describes how system should work

Communication | Simplicity | Feedback

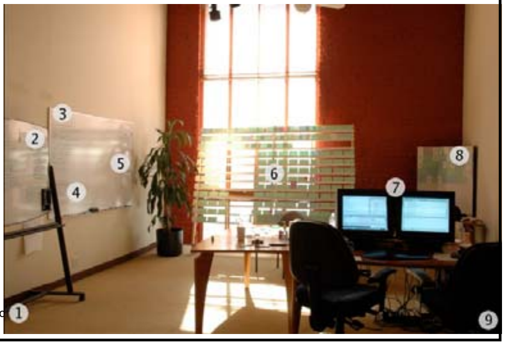
Courage

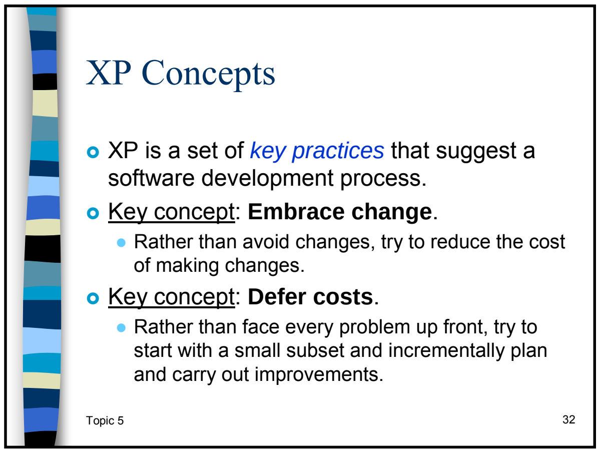
User Story / User Card

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



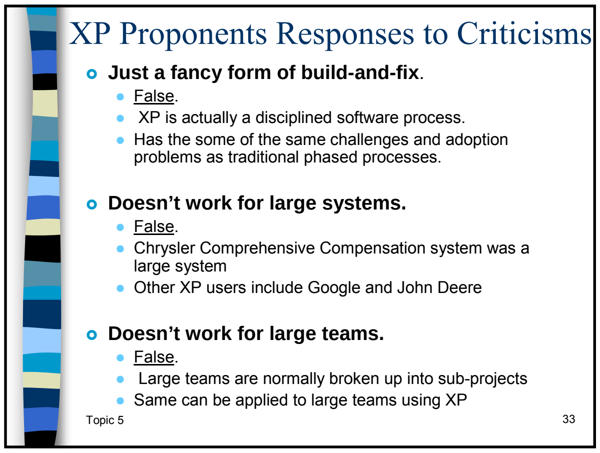
The XP Team Room





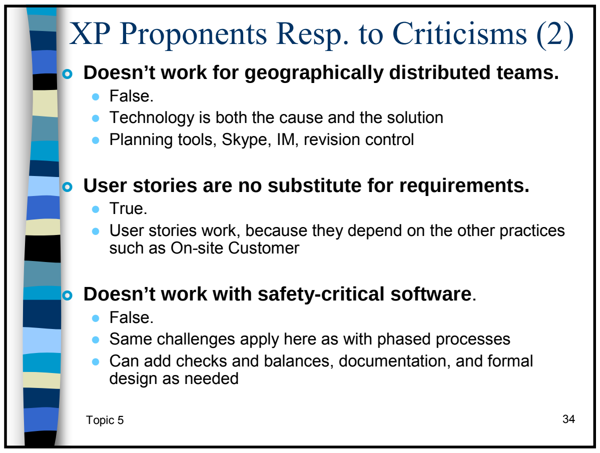
XP Concepts

- 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.



XP Proponents Responses to Criticisms

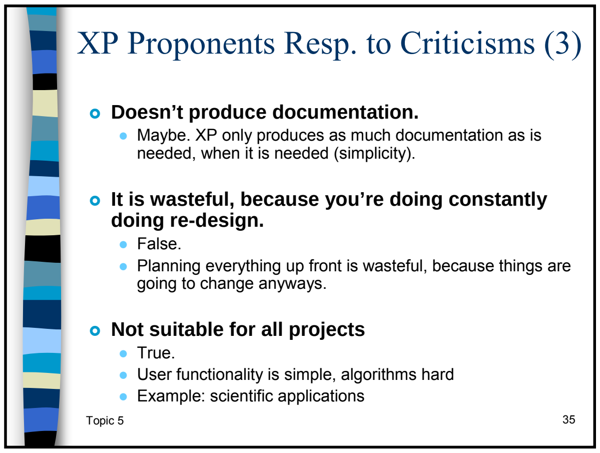
- **Just a fancy form of build-and-fix**.
  - False.
  - XP is actually a disciplined software process.
  - Has the some of the same challenges and adoption problems as traditional phased processes.
- **Doesn't work for large systems.**
  - False.
  - Chrysler Comprehensive Compensation system was a 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



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

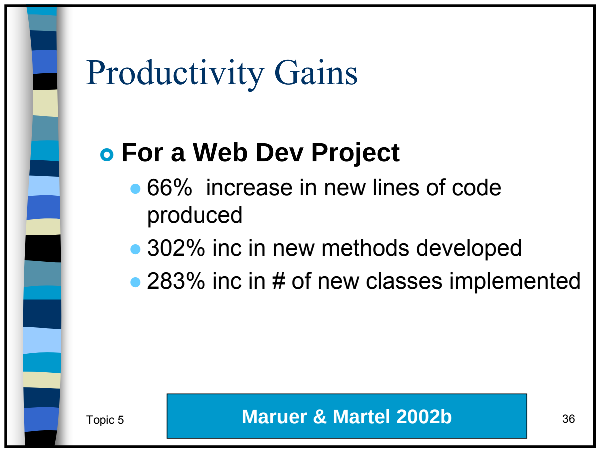
Topic 5 34



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

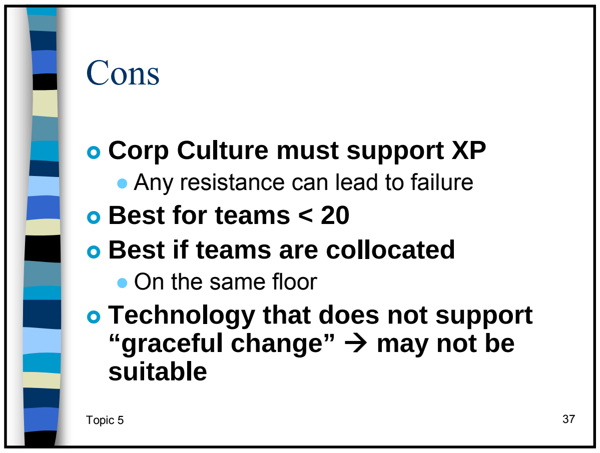
Topic 5 35



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

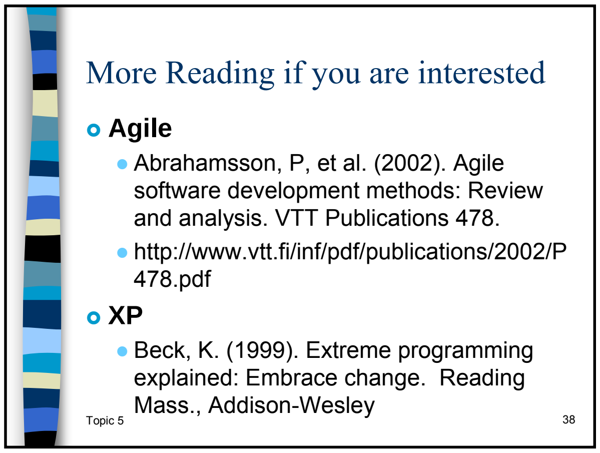
Topic 5 **Maruer & Martel 2002b** 36



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 5 37



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

Topic 5 38