

# Extreme Programming

CSE 308: Software Engineering

# What is Extreme Programming?

- [ Extreme Programming (XP) is a disciplined software methodology that fuses proven programming practices
  - these practices are used as thoroughly as possible
- [ “XP is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop software” (Beck)
  - “XP is an experiment in answer to the question, ‘How would you program if you had enough time?’” (Beck)

# Features of XP

- [ Early, concrete, continuing feedback from short cycles
- [ Incremental planning approach
- [ Flexible scheduling of the implementation of functionality
- [ Reliance on automated tests to monitor progress
- [ Reliance on tests and source code to communicate structure
- [ Evolutionary design process
- [ Close collaboration between developers

# Is This A New Idea?

- [ None of the ideas in XP are new
  - all of XP's practices have been proven over decades
- [ XP's innovation is:
  - putting all of these practices under one umbrella
  - using them thoroughly
  - making sure they support each other

# The Problem

# The Basic Problem

- [ Risk is the basic problem of software development
  - risk examples include schedule slip, high defect rates, false features, and staff turnover
- [ XP provides a style of software development that addresses these risks

<b>Risk</b>	<b>XP Response</b>
<b>Schedule slip</b>	<b>Short release cycles; prioritized features</b>
<b>High defect rate</b>	<b>Test-first approach</b>
<b>False features</b>	<b>Prioritized features</b>
<b>Staff turnover</b>	<b>Programmer responsibility; explicit code turnover</b>

# XP Development Cycle

- [ Pairs of programmers program together
- [ Development is driven by tests
  - “Until all the tests run, you aren’t done”
- [ Pairs evolve the design of the system
- [ Integration immediately follows development

# Software Development

— [ There are four basic variables in software development:

— cost

— time

— quality

— scope

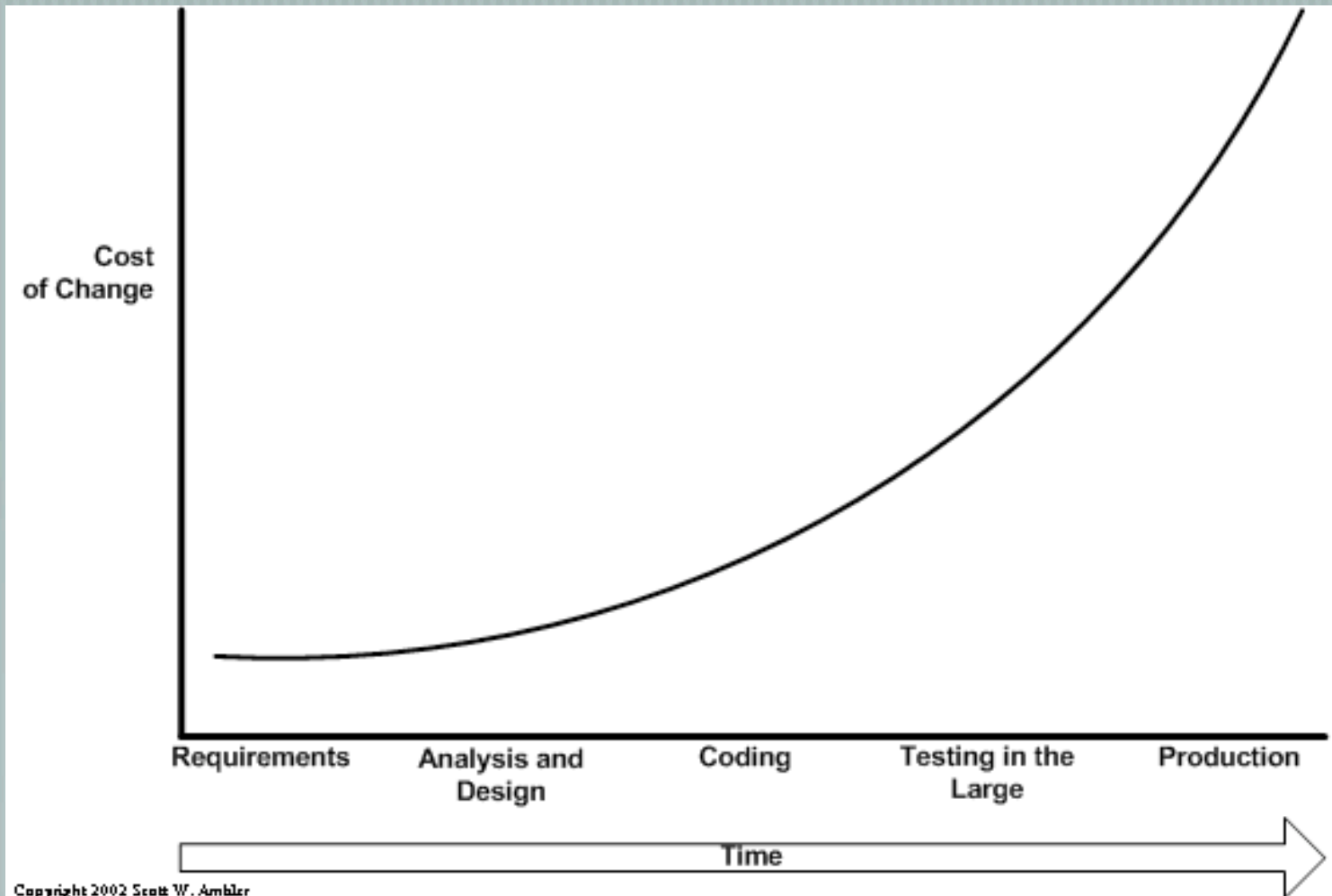
— [ External forces (customers, managers) get to pick 3

— [ Developers get to pick the value of the remaining variable

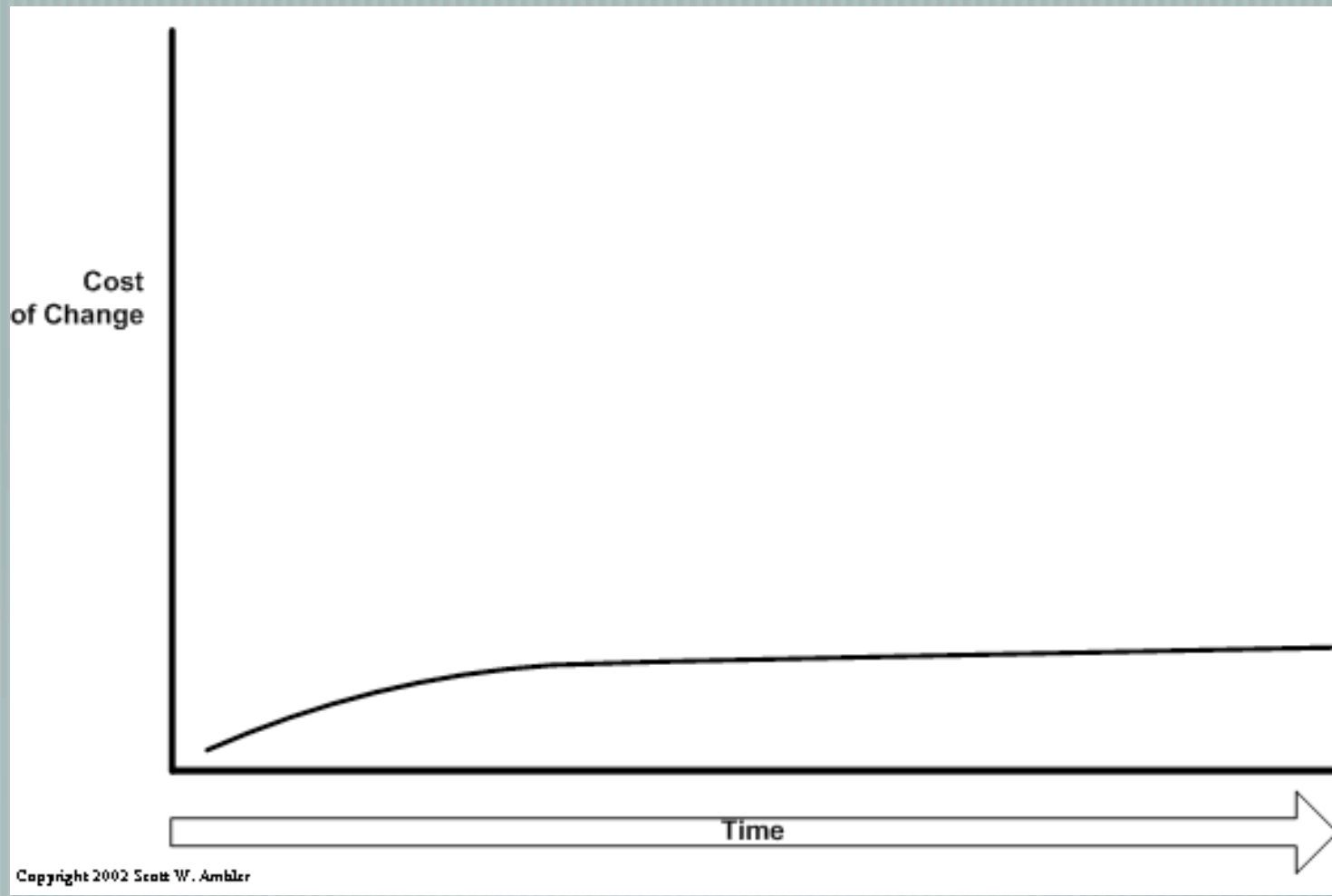
# Scope

- [ Scope is the most important variable
  - XP sees the “softness” of requirements as an asset
  - As development progresses, we adjust the scope
- [ XP avoids scope “creep” through estimation and prioritizing features

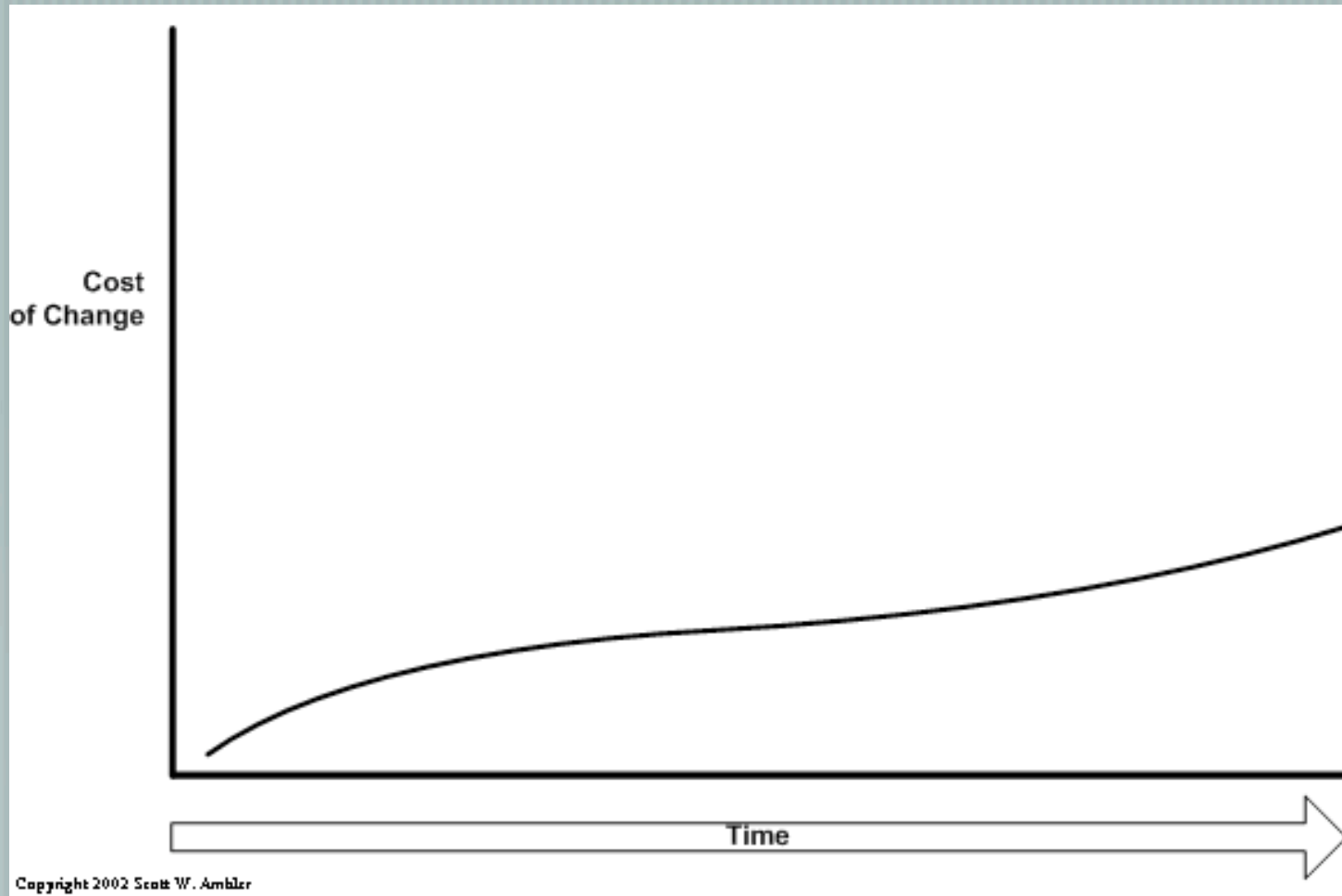
# The Cost of Change



# Beck's Version of The Graph



# A More Realistic Curve



# XP and the Cost of Change

- [ If the cost of change rose slowly over time, what would you do?
  - make big decisions as late as possible
  - only implement what you have to
  - introduce elements to the design only as they made things simpler

# The XP Paradigm

- [ Change is the only constant
- [ Always be prepared to move
- [ Make small constant corrections until you (eventually) reach your objective

# The Four Values of XP

- [ Communication
- [ Simplicity
- [ Feedback
- [ Courage

# Basic Principles of XP

- [ Rapid feedback
- [ Assume simplicity
- [ Incremental change
- [ Embracing change
- [ Quality work

# Less Central Principles

- Teach learning
- Small initial investment
- Play to win
- Concrete experiments
- Open, honest communication
- Work with instincts
- Accepted responsibility
- Local adaptation
- Travel light
- Honest measurement

# Basic Activities of XP

— [ XP is concerned with four basic activities:

— Coding

— Testing

— Listening

— Designing

# The Solution



# The Planning Game

<b>Business people decide on</b>	<b>Technical people decide on</b>
<b>Scope</b>	<b>Estimates</b>
<b>Priority</b>	<b>Consequences</b>
<b>Composition of releases</b>	<b>Process</b>
<b>Dates of releases</b>	<b>Detailed scheduling</b>

# Metaphor

- [ Each XP project is guided by an overarching metaphor
- [ The metaphor helps everyone understand the basic elements and their relationships
- [ This replaces what some people call “architecture”

# Simple Design

- [ The right design for the software at any given time:
  - runs all the tests
  - has no duplicated logic (or parallel class hierarchies)
  - states every intention important to the programmers
  - has the fewest possible classes and methods
- [ This is the **OPPOSITE** of “Design for tomorrow”

# Testing

- [ Programmers write unit tests
  - gives them confidence in the operation of the program
- [ Customers write functional tests
  - gives them confidence that the program satisfies its functional requirements

# Refactoring

- [ When implementing a program feature, ask, “Is there a simpler way this can be done?”
  - change the code to make it simpler to add a feature
- [ Refactor when the system asks you to
  - ex. when you have to duplicate code



www.dilbert.com  
scottadams@aol.com



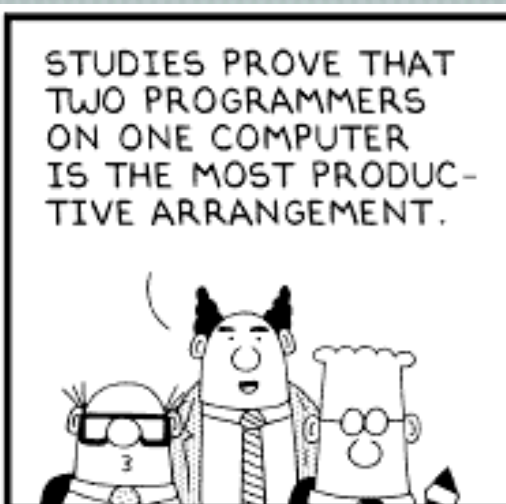
1/11/03 © 2002 United Feature Syndicate, Inc.



Copyright © 2003 United Feature Syndicate, Inc.



www.dilbert.com  
scottadams@aol.com



1/11/03 © 2002 United Feature Syndicate, Inc.



Copyright © 2003 United Feature Syndicate, Inc.

# Pair Programming

- [ All code is written with two people at the machine
- [ The partner with the keyboard/mouse focuses on implementation
- [ The other partner thinks strategically:
  - Is this approach going to work? Can we refactor?
- [ Pairing is dynamic; pairs change partners as needed

# Collective Ownership

- [ If you see an opportunity to improve a portion of code, you are required to do so
- [ Everyone takes responsibility for the whole system
- [ No one “owns” a particular piece of code

# Continuous Integration

- [ After a few hours (at most one day), new code is added to the system, one piece at a time
  - use a machine dedicated for this purpose
- [ Integration is not complete until the tests run 100%
  - if the tests fail, the newest code must be to blame

# 40-Hour Work Week

— [ Every programmer must be fresh, creative, and competent

— [ XP rule: you can't work a second week of overtime

— one week is okay; more means something is very wrong



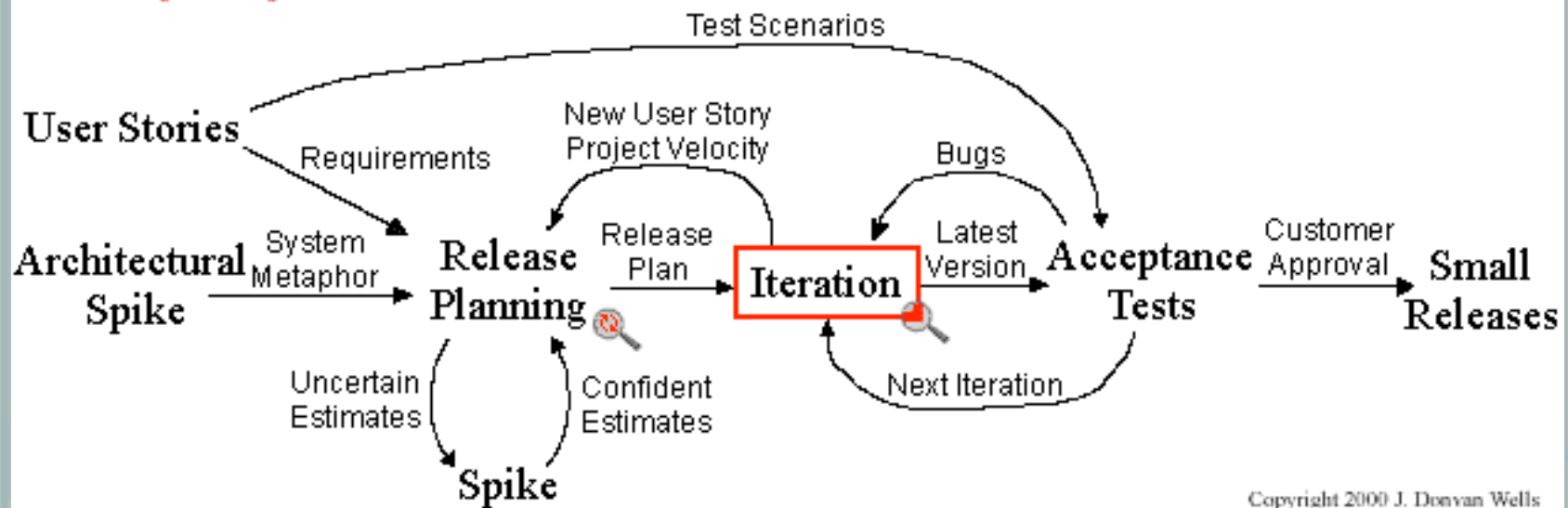
# On-Site Customer

- [ A real customer must sit with the coding team, available to answer questions
  - “customer” = “actual user of the finished system”
- [ Customers can do their own work at the same time
  - programmers rarely have 40 hours' worth of questions!

# Implementing XP



# Extreme Programming Project



Copyright 2000 J. Donovan Wells

# User Stories

- [ New/desired features are described by “user stories” that explain how they work



# Planning Strategy

- [ Do only the planning you need for the next horizon
  - ex. the next release, the end of the next iteration
- [ The person responsible for implementing gets to estimate
- [ Ignore dependencies between parts

# Design Strategy

- [ Small initial investment
- [ Assume simplicity
- [ Incremental change
- [ Travel light – don't overdesign!

# References

- [ Beck, K. “Extreme Programming Explained” (Addison-Wesley 2000)
- [ [www.xprogramming.com](http://www.xprogramming.com)
- [ [www.extremeprogramming.org](http://www.extremeprogramming.org)
- [ [www.agilemodeling.com](http://www.agilemodeling.com)

# Next Time

— [ Project Management with Scrum

— a simple practice for managing complex projects