

## Twelve Key Practices of XP

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

Week 2 Lecture Slide 1

## Programming Key Practices

- Simple design
  - Just do the simplest thing that will get you through this milestone
  - Eliminate duplication in the design
  - Don't over engineer, solve problems only when they occur
- Test-driven development
  - Write many simple, automated tests
  - Then, implement a feature and prove that it passes all tests
  - Re-run tests frequently to find defects and measure progress

Week 2 Lecture Slide 2

## More Programming Key Practices

---

- Refactoring
  - A change to the system that leaves its behavior unchanged, but enhances some nonfunctional quality—simplicity, flexibility, understandability, performance.
- Pair programming
  - Two people at a computer, with one keyboard, one monitor, one mouse
  - One codes, one thinks about the design and catches errors

Week 2 Lecture Slide 3

## More Programming Key Practices

---

- Collective code ownership
  - Any developer can change any code any time
  - Everyone codes to the same stylistic standards
  - Automated tests catch any errors that are introduced
- Continuous integration
  - Merge changes from different developers very frequently
- Coding standards
  - Everyone must use the same coding standards
  - Corollary to collective code ownership

Week 2 Lecture Slide 4

## Management Key Practices

---

- The Planning Game
  - Stakeholders specify what the system needs to do
    - Customer provides user stories and priorities
    - User stories recorded on index cards
  - Development specifies how much each feature costs and what budget is available per day/week/month
- Small releases
  - Build the smallest, simplest system possible
  - Then make a long series of small upgrades, each of which also works
- 40-hour work week
  - It is better to be awake and alert
  - Don't work overtime two weeks in a row

Week 2 Lecture Slide 5

## Customer Key Practices

---

- On-site customer
  - Need customer/user around to answer questions
  - Builds a bond, working relationship
- Metaphor
  - A common story for business and technical people to understand the goal
  - Standard names, pictures
  - Like architecture, but purpose is communication

Week 2 Lecture Slide 6

## Comments on XP

---

- XP is better than \_no\_ process at all.
- XP is closest to build-and-fix process model because planning and doing are both incremental with the same duration.
- Certain practices of XP are good:
  - User stories are a quick and easy way to get started capturing user needs
  - Continuous integration, frequent iterations and regression tests
  - Joint code ownership, shared coding style

Week 2 Lecture Slide 7

## Criticisms of XP

---

- Some practices of XP are not good:
  - Short-sighted planning is wasteful
  - Tests are no substitute for written requirements
  - Code is the central artifact, there are few formal requirements or design documents. That means less effort so long as all developers remember the history of decision-making, but it makes it very hard for new developers to join the project later.
- Not suitable for all projects
  - Geographically distributed teams
  - User functionality is simple, algorithms hard
    - Example: scientific applications

Week 2 Lecture Slide 8

## Agile Manifesto

---

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

Week 2 Lecture Slide 9

## Lab 1: Pair Programming

---

- Pair programming is the most talked-about aspect of XP
  - Shock value and cachet
- Pair programming is not:
  - One person typing and the other person watching
  - Tutoring
  - Fixed- pairs
- Pair programming is:
  - Two people working together as a team
  - Person typing more concerned with current problem, while partner is thinking about big picture and helping to catch errors

Week 2 Lecture Slide 10

## Lab 1: Pair Programming

---

- Who gets to drive the computer?
  - Person who is less skilled
    - Less familiar with big picture issues
    - Allows other person more time to think
    - Better for joint understanding of problem
  - The control freak
  - Person who is more skilled
    - More educational for buddy (who still needs to be actively participating, by pointing out errors, asking questions)
- Benefits
  - Communication and education
  - Higher quality code
  - More than one person familiar with each part of code
  - Leveraging strengths and weaknesses

Week 2 Lecture Slide 11

## Lab 1: Pair Programming

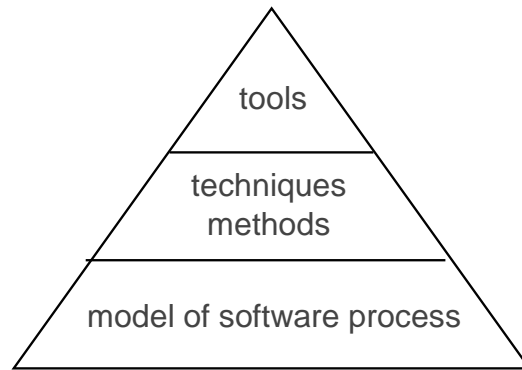
---

- Choose your partner carefully
  - Particularly important for people who are more introverted
- Contribute even if you aren't driving
  - You might be asked to take over driving at any time!
- Actively listen to your partner

Week 2 Lecture Slide 12

## Selecting a Tool

---



Week 2 Lecture Slide 13

## Conflict: Practice and Problems

---

- Why is there a gap between actual practice and state of the art?
  - Gap exists in both student projects and industry
- Focus on the end product, not how you get there.
  - Don't have time to learn a new tool or method when you have an assignment due.
  - Tools have a steep learning curve.

Week 2 Lecture Slide 14

## Learning Curve

---

- It takes  $\geq 10,000$  hours to become an expert
- A software engineer needs to know:
  - Programming languages
  - Programming techniques
  - Software tools
  - Methods
  - Notations
  - Domain knowledge
  - Local knowledge
- 40 hours/week times 50 weeks/year = 2000 hours/year
  - Takes 5 years to become an experienced SE

Week 2 Lecture Slide 15

## Analogy

---

- Cello = tool
- How to play = method
- Sheet music = notation

Week 2 Lecture Slide 16

## Summary

---

- Gap between personal software development practices and those needed for software engineering
- Tend to focus on the *what* instead of *how*
- Steep learning curve for tools, methods, and notations
- Takes 10,000 hours to become an expert
  
- This course will help with the gap.
  - Focus on know-how, instead of know-what
  - Already becoming expert software engineers

Week 2 Lecture Slide 17

## Definitions for Software Tools

---

- A **software tool** supports a specific task in the software development process
  - Example: a compiler, model checker
- A **workbench** is a set of tools with a limited scope
- An **environment** supports an entire development process

Week 2 Lecture Slide 18

## Classification for Software Tools

Dimension	Typical Values
Breadth of support	Tool, workbench, environment, ...
Class of problem	Embedded, business, real-time,...
Size of system	Small, medium, large
User scale	Individual, family, city, or state
Process scale	Product, people, or product-and-people
Process support	None, fixed, or variable
Execution paradigm	State machine, Petri net, production rules, procedures,...

Week 2 Lecture Slide 19

## Productivity Rates

- Longhorn Project (2003)
  - 16 MLOC, 5000 people, 3 years
  - 1067 LOC/person/year
- Grady and Caswell at HP (1987)
  - ~1100 LOC/person/year
- Brooks (1975)
  - IBM OS/360
    - 600-800 instructions/person/year in control group
    - 2000-3000 instructions/person/year in language translator group
  - Corbató's Data
    - 1200 lines/person/year (PL/I)

Week 2 Lecture Slide 20

## Factors Affecting Productivity Rates

---

- Application domain experience
- Process quality
- Project size
  - Negative relationship
- Technology support
- Working environment

Week 2 Lecture Slide 21

## How are project plans created?

---

- A wish list for the project is created
  - Clients, executives, product managers, and programmers have input
- Tasks on the wish list are sized
  - Programmers are asked about feasibility and effort required- they give their best guess
- Numbers are passed up the chain
  - Numbers are inflated and deflated to suit whether there is available:
    - Money
    - Calendar time, work time
    - Market pressures, e.g. competitive bids, competitor time to market, trade shows
- Project plans are based on effort estimates!

Week 2 Lecture Slide 22

## Poor Estimation Techniques

---

- Guessing
- Parkinson's Law
- Pricing to win
- Budget method
  
- Brooks, Chapter 2
  - “Good cooking takes time. If you are made to wait, it is to serve you better, and to please you.”
  - Gutless estimating

Week 2 Lecture Slide 23

## Parkinson's Law

---

- “Work expands to fill the time available.”
- The project takes all the available time
  - Adjust functionality?
  
- Advantage
  - No overspending
- Disadvantages
  - Unethical
  - Unreliable
  - System is usually unfinished

Week 2 Lecture Slide 24

## Pricing to win

---

- The project costs less than whatever our competitors bid
- Advantages
  - You get the contract
- Disadvantages
  - Unethical
  - Unreliable
  - The probability that the customer gets the system he or she wants is small.
  - Costs do not accurately reflect the work required

Week 2 Lecture Slide 25

## Pricing to win

---

- When detailed information is lacking, it may be the only appropriate strategy
- The project cost is agreed on the basis of an outline proposal and the development is constrained by that cost
- A detailed specification may be negotiated or an evolutionary approach used for system development

Week 2 Lecture Slide 26

## Budget Method

---

- Similar to Parkinson's law, but based on money instead of time
- The project costs whatever the customer has to spend on it
- Advantages and Disadvantages similar to Parkinson's Law

Week 2 Lecture Slide 27

## Better Estimation Techniques

---

- Based on experience or hard data
- Expert judgment
- Estimation by analogy
- Variation: Delphi method
- Algorithmic cost modeling

Week 2 Lecture Slide 28

## Expert judgement

---

- One or more experts in both software development and the application domain use their experience to predict software costs.
- Advantages
  - Relatively cheap estimation method.
  - Can be accurate if experts have direct experience with similar systems
- Disadvantages
  - Very inaccurate if there are no experts.
  - Does not use hard data

Week 2 Lecture Slide 29

## Estimation by Analogy

---

- The cost of a project is estimated by comparing the project to a similar project in the same application domain
- Advantages
  - Accurate if comparable project data available
- Disadvantages
  - Impossible if no comparable project has been undertaken.
  - Estimates can be inaccurate if details overlooked.
  - Subsequent similar projects can be quicker.

Week 2 Lecture Slide 30

## Delphi Method

---

- Idea: Create a group expert opinion, while counterbalancing personality factors in process
  - Group of expert estimators + moderator
1. Experts independently create estimates.
  2. Moderator collects written estimates from individuals.
  3. Estimates are distributed to group.
    - No names
  4. Experts deliver new estimates based on new information from moderator.
  5. Continue until consensus is reached.

Week 2 Lecture Slide 31

## Algorithmic Cost Modeling

---

- Cost and development time for a project is estimated from an equation
- Equations can come from research or industry
  - Work best if they are tailored using personal and organizational data

Week 2 Lecture Slide 32

## Basic Equation

---

$$E = (a + S^c)m(X)$$

- S is estimated size of the systems (LOC)
- a, c, m, are constants
  - a is an organization-dependent constant
  - c reflects the disproportionate effort for large projects
  - m is a multiplier reflecting product, process and people attributes
- X is a vector of cost factors,  $x_1 \dots x_n$ 
  - Complexity of project, risk, resources, methods
- Most commonly used product attribute for cost estimation is code size
- Most models are basically similar but with different values for A, B and M

Week 2 Lecture Slide 33

## Problems with Algorithmic Estimation

---

- Effort estimates are based on size
  - Highly inaccurate at start of project
- Size is usually given in lines of code
- Lines of code does not reflect the difficulty
  - Some short programs are harder to write than long ones
  - Lines of code  $\neq$  effort
    - Not all activities produce code
  - Programming Language: Java vs. assembler
- Recall Brooks Chapter 2
  - Effort  $\neq$  progress
  - The B exponent is an attempt to account for communication and complexity costs, but basic problem remains

Week 2 Lecture Slide 34

## Data Collection

---

- Regardless of the method or model used, data is needed for calibration
- Programmers need to know their own “constant adjustment factors”
  - Goal of Personal Software Process to establish such a database

Week 2 Lecture Slide 35

## So, what can I do?

---

- You
  - Don't have a historical database
  - Are not an expert
- Generate estimates using multiple models and compare based on your guesses or assumptions
  - Similar to using the models as your personal experts in Delphi method
  - Candidate models:
    - Walston and Felix (simple and easy to use)
    - COCOMO 2 (complicated and detailed)
    - DeMarco (based on UI requirements)
    - Brooks, p. 20
      - 1/3 planning, 1/6 coding, 1/4 component tests and early system test, 1/4 system test

Week 2 Lecture Slide 36