Software Development Processes: *Extreme Programming*

Software Engineering

Millersville University

Extreme Programming

- Waterfall model inspired by civil engineering
- Civil engineering metaphor is not perfect
  - Software is more organic than concrete
  - You “grow the software” to meet changing requirements
- Extreme Programming (XP) addresses this
  - A version of the iterative model discussed before

Goals

- Minimize unnecessary work
- Maximize communication and feedback
- Make sure that developers do most important work
- Make system flexible, ready to meet any change in requirements

History

- Kent Beck
    - Influential book “Extreme Programming Explained” (1999)
- Speed to market, rapidly changing requirements
- Some ideas go back much further
    - “Test first development” used in NASA in the 60s

XP Practices

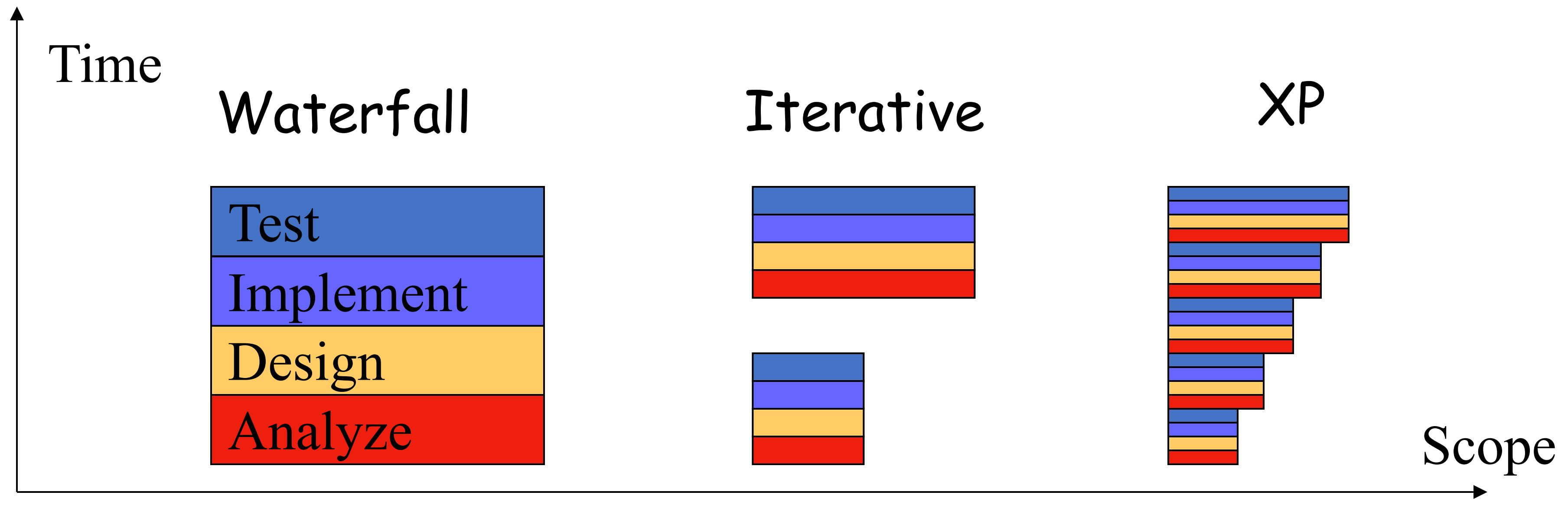
- On-site customer
- The Planning Game
- Small releases
- Testing
- Simple design
- Refactoring
- Metaphor
- Pair programming
- Collective ownership
- Continuous integration
- 40-hour week
- Coding standards

XP Process (2-3 week cycle)

1. Meet with client
   - User stories + acceptance tests
2. Planning game
   - Break stories into tasks, estimate cost
   - Client prioritizes stories to do first
3. Implementation
   - Write programmer tests first
   - Simplest possible design to pass the tests
   - Code in pairs
   - Occasionally refactor the code
4. Evaluate progress and Reiterate

Extreme Programming (XP)

- XP: like iterative but taken to the *extreme*



XP Customer

Expert customer is part of the team

- On site, available constantly
- XP principles: communication and feedback
- Make sure we build what the client wants

Customer involved active in all stages:

- Clarifies the requirements
- Negotiates with the team what to do next
- Writes and runs acceptance tests
- Constantly evaluates intermediate versions
- Question: How often is this feasible?

Example: Accounting Customer Tests

- Tests are associated with (one or more) stories

1. If I create an account “savings”, then another called “checking”, and I ask for the list of accounts I must obtain: “checking”, “savings”
2. If I now try to create “checking” again, I get an error
3. If now I query the balance of “checking”, I must get 0.
4. If I try to delete “stocks”, I get an error
5. If I delete “checking”, it should not appear in the new listing of accounts

…

Automate Acceptance Tests

Customer can write

(and later rerun tests)

E.g., customer writes an XML table with data examples, developers write tool to interpret table

Tests should be automated

To ensure they are run after each release

Tasks

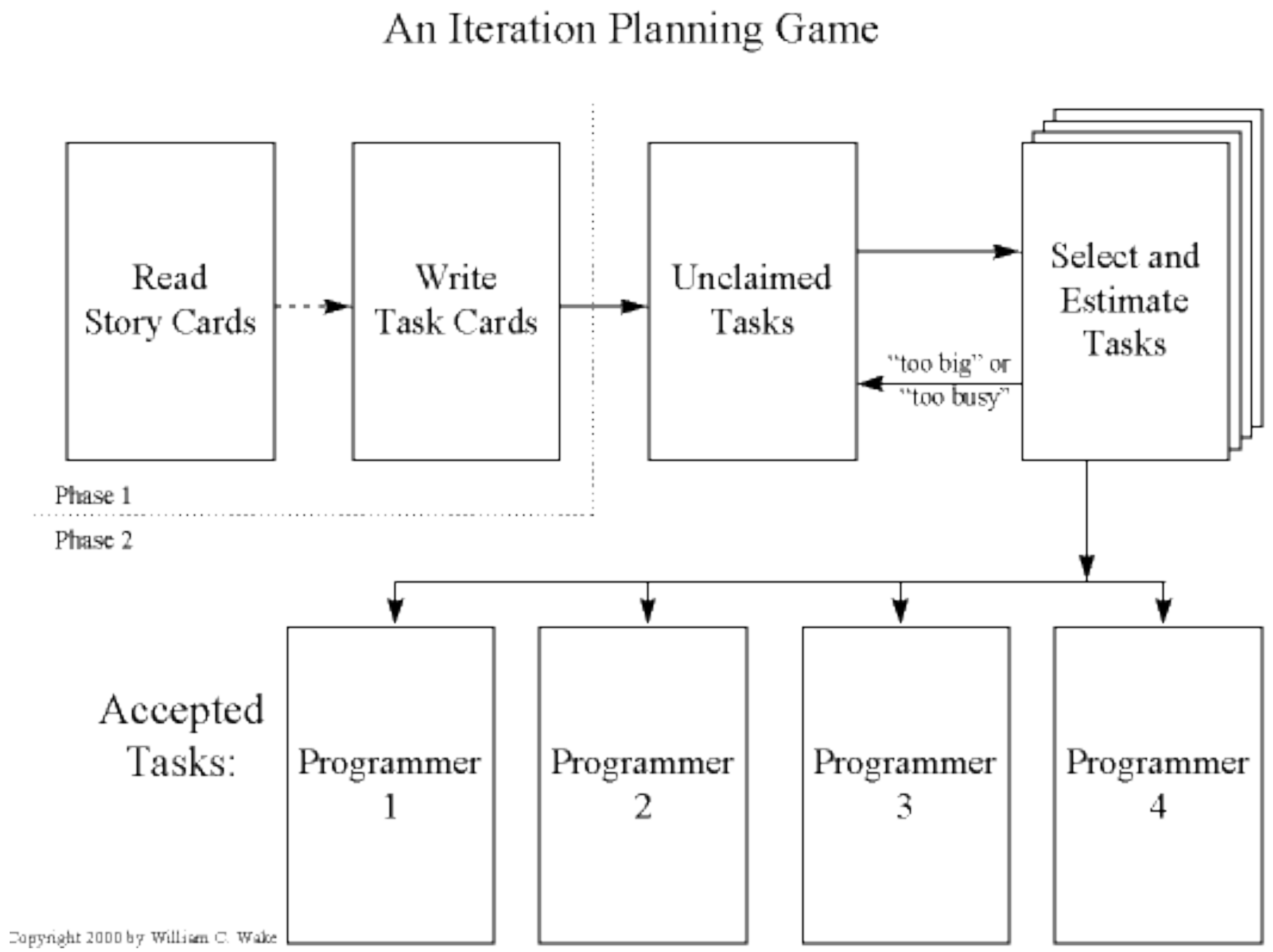
- Each story is broken into tasks
    - To split the work and to improve cost estimates
- Story: customer-centered description
- Task: developer-centered description
- Example:
    - Story: “I can create named accounts”
    - Tasks: “ask the user the name of the account”
      “check to see if the account already exists”
      “create an empty account”

- Break down only as much as needed to estimate cost
- Validate the breakdown of stories into tasks with the customer

Tasks

- If a story has too many tasks: break it down
- Team assigns cost to tasks
  - We care about relative cost of task/stories
  - Use abstract “units” (as opposed to hours, days)
  - Decide what is the smallest task, and assign it 1 unit
  - Experience will tell us how much a unit is
  - Developers can assign/estimate units by bidding: “I can do this task in 2 units”

Play the Planning Game



Planning Game

- Customer chooses the important stories for the next release
- Development team bids on tasks
  - After first iteration, we know the speed (units/week) for each subteam
- Pick tasks => find completion date
- Pick completion date, pick stories until you fill the budget
- Customer might have to re-prioritize stories

XP: Pair programming

- Pilot and copilot metaphor
  - Or driver and navigator
- Pilot types, copilot monitors high-level issues
  - simplicity, integration with other components, assumptions being made implicitly
- Disagreements point early to design problems
- Pairs are shuffled periodically

Pair programming

WE'RE GOING TO TRY SOMETHING CALLED EXTREME PROGRAM-MING.
scottadams@aol.com
www.dilbert.com
FIRST, PICK A PARTNER. THE TWO OF YOU WILL WORK AT ONE COMPUTER FOR FORTY HOURS A WEEK.
© 2002 United Feature Syndicate, Inc.
1/9/03
THE NEW SYSTEM IS A MINUTE OLD AND I ALREADY HATE EVERYONE.

Benefits of Pair Programming

- Results in better code
  - instant and complete and pleasant code review
  - copilot can think about big-picture
- Reduces risk
  - collective understanding of design/code
- Improves focus and productivity
  - instant source of advice
- Knowledge and skill migration
  - good habits spread

Why Some Programmers Resist Pairing ?

- “Will slow me down”
  - Even the best hacker can learn something from even the lowliest programmer
- Afraid to show you are not a genius
  - Neither is your partner
  - Best way to learn

Why Some Managers Resist Pairing?

- Myth: Inefficient use of personnel
  - That would be true if the most time consuming part of programming was typing !
  - 15% increase in dev. cost, and same decrease in bugs
- Resistance from developers
  - Ask them to experiment for a short time
  - Find people who want to pair

Evaluation and Planning

- Run acceptance tests
- Assess what was completed
  - How many stories ?
- Discuss problems that came up
  - Both technical and team issues
- Compute the speed of the team
- Re-estimate remaining user stories
- Plan with the client next iteration

What’s Different About XP

- No specialized analysts, architects, programmers, testers, and integrators
  - every XP programmer participates in all of these critical activities every day.
- No complete up-front analysis and design
  - start with a quick analysis of the system
  - team continues to make analysis and design decisions throughout development.

What’s Different About XP

- Develop infrastructure and frameworks as you develop your application
    - not up-front
    - quickly delivering business value is the driver of XP projects.

When to (Not) Use XP

- Use for:
  - A dynamic project done in small teams (2-10 people)
  - Projects with requirements prone to change
  - Have a customer available
- Do not use when:
  - Requirements are truly known and fixed
  - Cost of late changes is very high
  - Your customer is not available (e.g., space probe)

What can go wrong?

- Requirements defined incrementally
  - Can lead to rework or scope creep
- Design is on the fly
  - Can lead to significant redesign
- Customer representative
  - Single point of failure
  - Frequent meetings can be costly

Conclusion: XP

Extreme Programming is an incremental software process designed to cope with change

With XP you never miss a deadline; you just deliver less content