

Software Process

Abram Hindle
hindle1@ualberta.ca

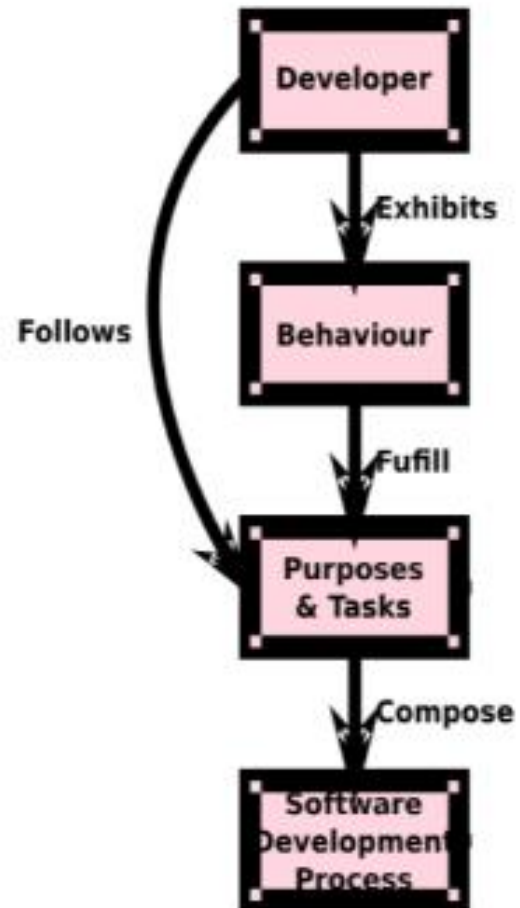
Henry Tang
hktang@ualberta.ca

Department of Computing Science
University of Alberta

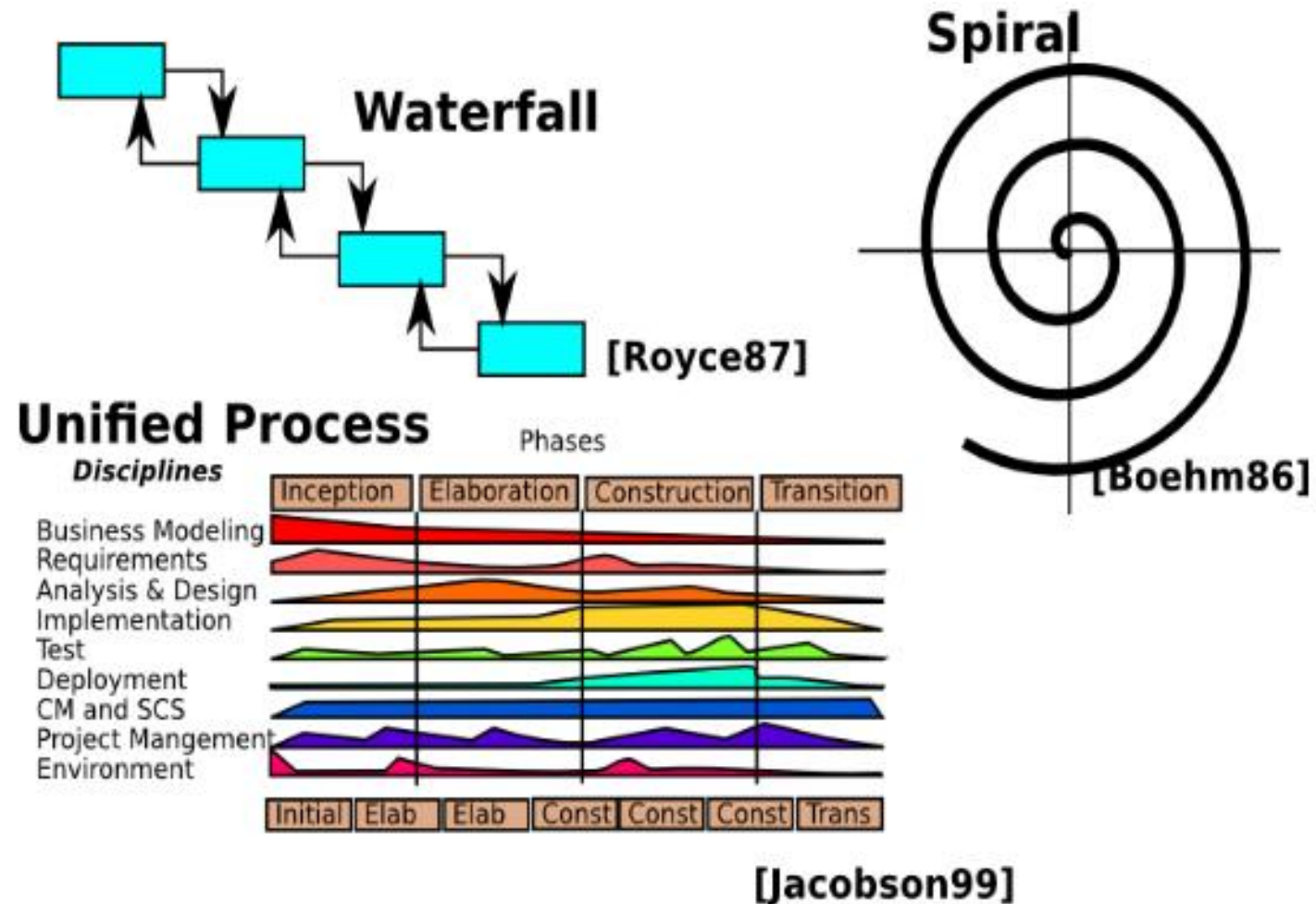
CMPUT 301 – Introduction to Software Engineering
Slides adapted from Dr. Hazel Campbell, Dr. Ken Wong



What Makes a Process?



Software Development Processes



Developer Perspective

- Software engineering:
 - Manage complexity, scale, lifetime
 - Increase quality
 - Reduce defects
 - Reduce maintenance and support costs
 - Reduce time-to-market
 - Reuse successful solutions
 - Apply methods and tools
 - Iterate and optimize

User Perspective

- Software usability:
 - Meets needs
 - Increase productivity
 - Easy to learn
 - Effective to use
 - Reduce errors
 - Safe to use

User Perspective

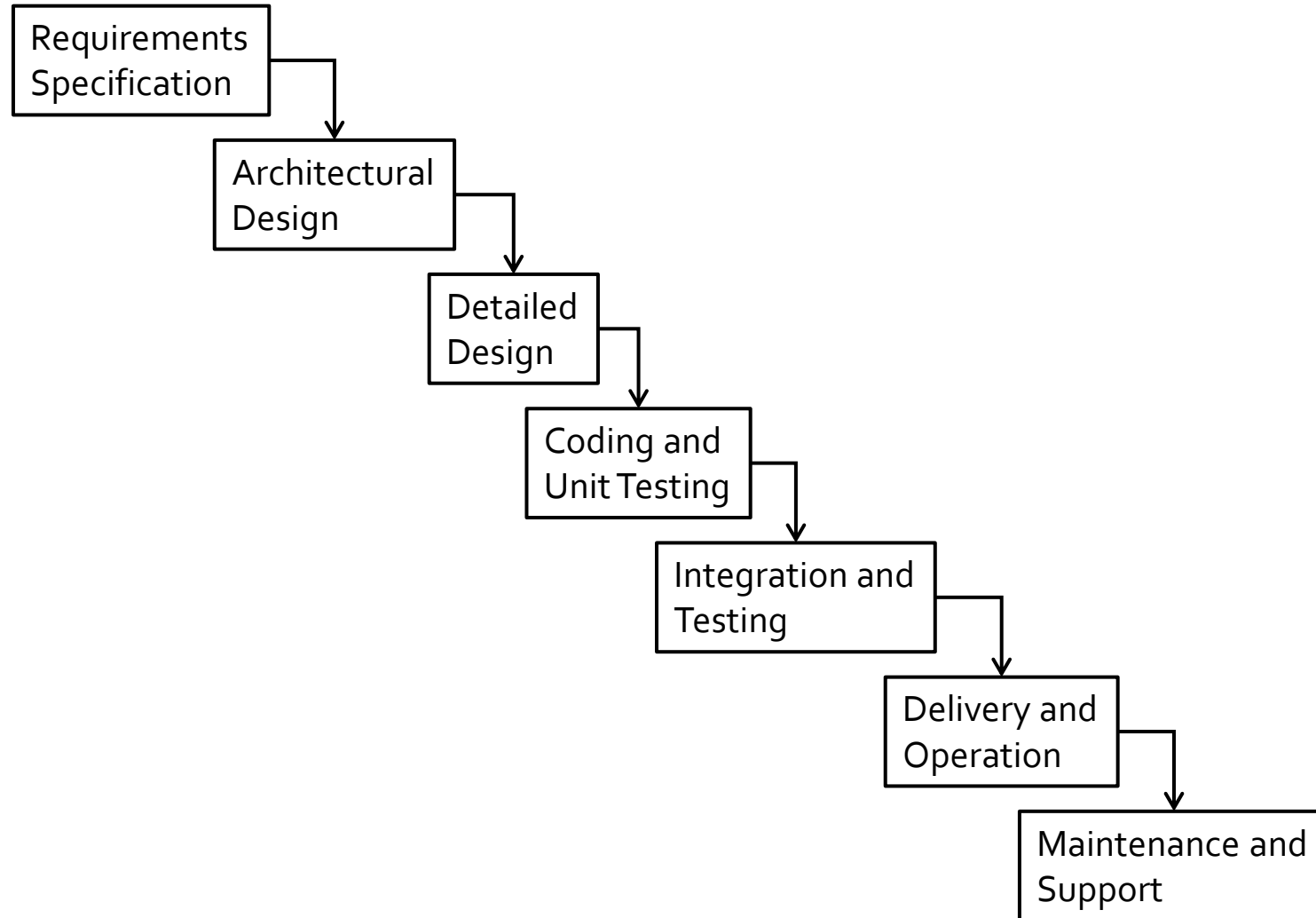
- Experience:
 - Satisfying
 - Motivating
 - Looks nice
 - Enjoyable
 - Fun

Meeting Needs

- Verification
 - Making sure you develop the *system right* (i.e., according to the requirements)

Waterfall

Waterfall Lifecycle Model



Discussion

- What are some pros and cons of the waterfall model?

Waterfall

- Pros:
 - Easily understood
 - Enforces discipline
 - Verification at every phase
 - Well-documented product

Waterfall

- Cons:
 - Uses a manufacturing view of software
 - Most software is not made as a “final” product
 - Customer must be patient
 - But time-to-market is critical
 - Customer sees the system only at the end
 - May not satisfy their real needs
 - No early feedback

Waterfall

- Cons:
 - Dependence on requirements being “right” at the start
 - This is almost never the case
 - Could end up building the wrong system
 - Requirements must all be known up front
 - But cannot always foresee all the necessary and changing requirements
- Summary
 - Need to be able to iterate – waterfall is not effective

Prototyping

Meeting Needs

- Validation
 - Making sure you develop the *right system* (i.e., what the customer really needed)

Prototyping

- Iterative design:
 - Cycling through several designs
 - Improving the product with each pass
- Various approaches (in combination):
 - Throwaway
 - Incremental
 - Evolutionary

Throwaway Prototyping

- Process:
 - Build and test prototype
 - Gain knowledge for the real product
 - What is necessary
 - What works
 - What does not work
 - “Throw away” the prototype, then “develop” the product for real

Throwaway Prototyping

- Pros:
 - More communication between users and developers
 - Functionality is introduced earlier, which is good for morale

Throwaway Prototyping

- Cons:
 - Building the prototype must be rapid
 - Some qualities may be sacrificed, like security, reliability, etc.
 - Temptation to use the throwaway prototype in the final product

Incremental Prototyping

- Process:
 - Triage system into separate “increments”
 - I.e., “must do”, “should do”, “could do”
 - Develop and add one increment at a time
- Example: Accounting system
 - Prototype 1 – general ledger
 - Prototype 2 – accounts receivable/payable
 - Prototype 3 – payroll

Evolutionary Prototyping

- Process:
 - Feature is refined or “evolved” over time
- Example: Text editor
 - Prototype 1 – keyboard Cut and Paste
 - Prototype 2 – touchscreen Cut and Paste
 - Prototype 3 – Cut and Paste works with Undo

Other Kinds of Prototypes

- User interface sketches
 - Hand-drawn or using drawing tool
- Storyboards
 - Graphical depiction of user interface
 - Like a comic strip, but only draw the UI

Other Kinds of Prototypes

- Index cards, Post-It[®] notes
 - E.g., tasks in a project plan
 - E.g., classes in an object-oriented analysis
 - E.g., pages in a web site structure

Other Kinds of Prototypes

- Physical mockups:
 - E.g., made from wood, clay, or foam



Balsa wood mock-up



Partial clay mock-up



Precision mock-up

© Canon



© Alan Kay

Other Kinds of Prototypes

- Wizard of Oz:
 - “Pay no attention to that man behind the curtain!”
 - Feature is “implemented” through human intervention “behind the scenes”

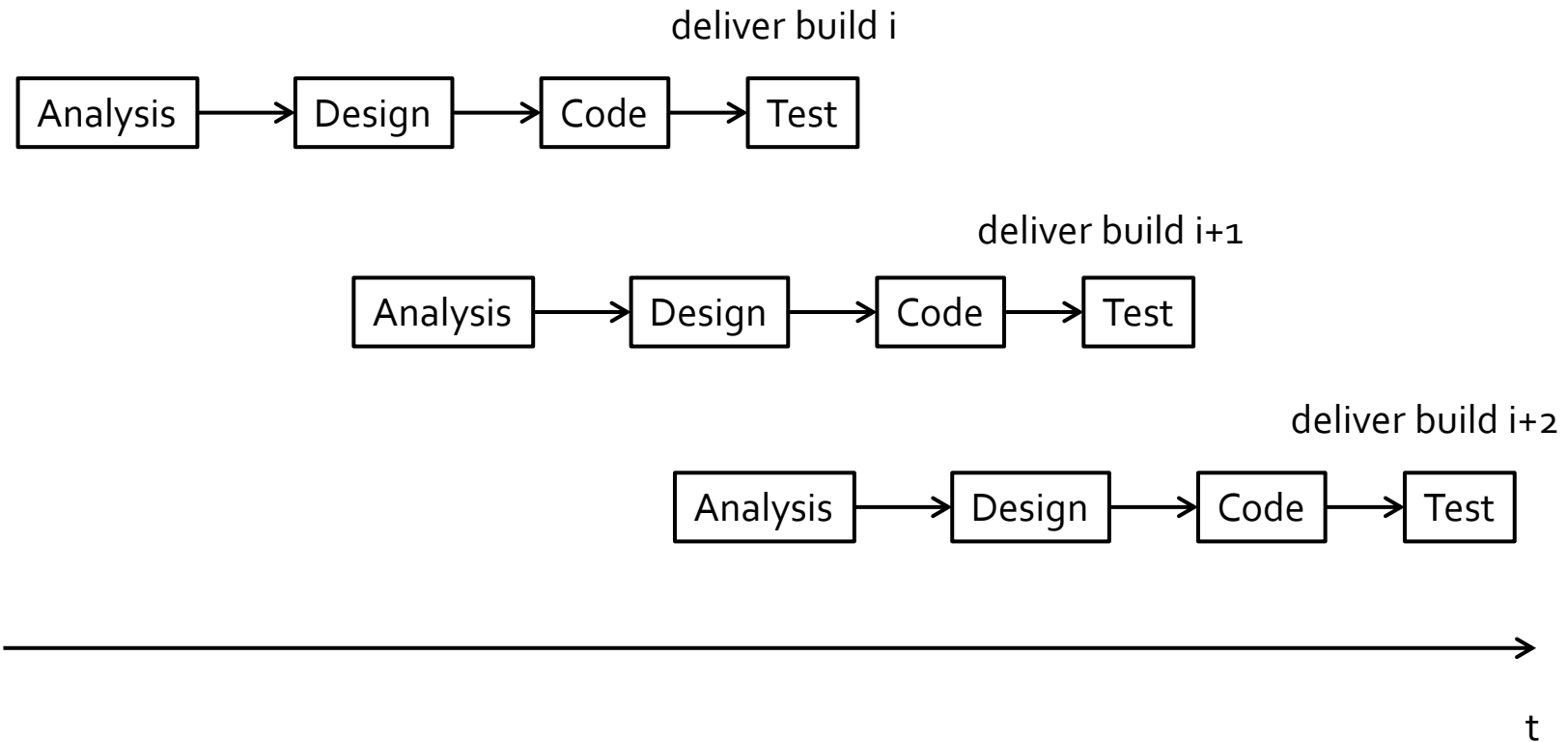


Staged Delivery

Staged Delivery

- Developers:
 - Deliver the system in a series of working releases or builds
- Users:
 - Use some functionality while the rest continues to be developed
- Possible parallelism:
 - Production and development systems
 - Staggered development streams

Staggered Builds



Staged Delivery

- Pros:
 - Provides more options
 - Different builds focus on specific features
 - Reduces estimation errors
 - Risks are reduced earlier

Staged Delivery

- Cons:
 - Overhead needed to plan and drive the product toward staged releases
 - Extra complexity of supporting multiple versions in the field

Agile Practices

“Agile Manifesto”

- <http://agilemanifesto.org/>

Agile Principles

- Individuals and interactions
- Working software
- Customer collaboration
- Responding to change

Agile Principles

- Individuals and interactions:
 - Trust motivated individuals
 - Face-to-face conversation
 - Best work emerges from self-organizing teams
 - Team reflects on and adjusts their behavior
 - Promote constant, sustainable pace

Agile Principles

- Working software:
 - The main measure of progress
 - Continuous, frequent delivery of value

Agile Principles

- Customer collaboration:
 - Customers and developers work together
 - Satisfy customer early

Agile Principles

- Responding to change:
 - Welcome changing requirements, even if late
 - Technical excellence and good design
 - Simplicity – art of maximizing work not done

Prioritizing Stories (Features)

- High priority
 - **Must** be done
 - Complete first
 - Risk level:
 - Will cause big problems if not done (first)
 - Will cause big problems if it breaks

Prioritizing Stories (Features)

- Medium priority
 - **Should** be done
 - Complete second
 - Risk level:
 - Will cause some problems if not done (before other user stories)
 - Will cause some problems if it breaks

Prioritizing Stories (Features)

- Low priority
 - **Could** be done
 - Complete third
 - Risk level:
 - Only minor problems if not done (before other user stories)
 - Only minor problems if it breaks

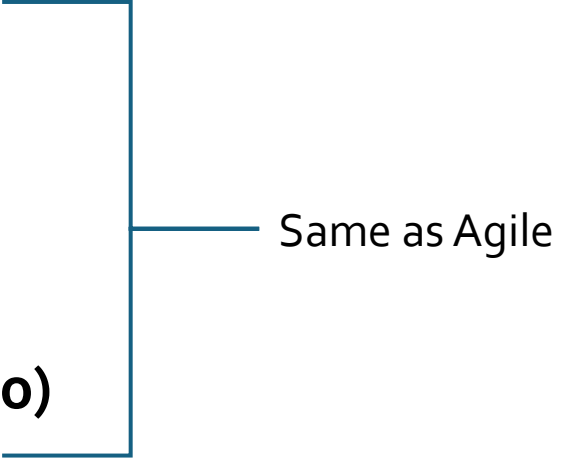
Prioritizing Stories (Features)

- No priority
 - Nice to have but not needed
 - Do it last
 - Risk level:
 - No problems if not done (before other user stories)

eXtreme Programming (XP)

- <http://www.extremeprogramming.org/>
- Predecessor to Agile

XP

- Philosophy:
 - **Communication**
 - **Feedback**
 - **Simplicity**
 - **Programmer friendly**
 - **For small teams (up to about 20)**
 - Code-centric
 - Requires courage
- 
- Same as Agile

XP

- 12 practices:
 - 40-hour week
 - Metaphor
 - Simple design
 - Collective ownership
 - Coding standards
 - Small releases
 - Continuous integration
 - Refactoring
 - Planning game
 - Testing
 - On-site customer
 - Pair programming

XP

- For programmer welfare:
 - 40-hour week
 - Work no more than 40 hours a week
 - Never work overtime a second week in a row

XP

- For shared understanding:
 - Metaphor
 - Guide development with a shared story of how the system works
 - Simple design
 - Design the system as simply as possible; remove extra complexity when discovered

XP

- For shared development:
 - Collective ownership
 - Anyone can change any code anywhere in the system at any time
 - Coding standards
 - Write all code according to rules that enhance communication and understanding through code

XP

- For continuity:
 - Small releases
 - Put simple system into production quickly, then release new versions on a very short cycle
 - Continuous integration
 - Integrate and build the system many times a day
 - Refactoring
 - Restructure the system to improve its design, simplicity, or flexibility

XP

- For feedback:
 - Planning game
 - Determine scope of the next iteration and overall release together with customer
 - Testing
 - Write automated unit tests first before the code; customer writes tests in requirements
 - On-site customer
 - Include a real, live user on the team, available full-time to answer questions quickly

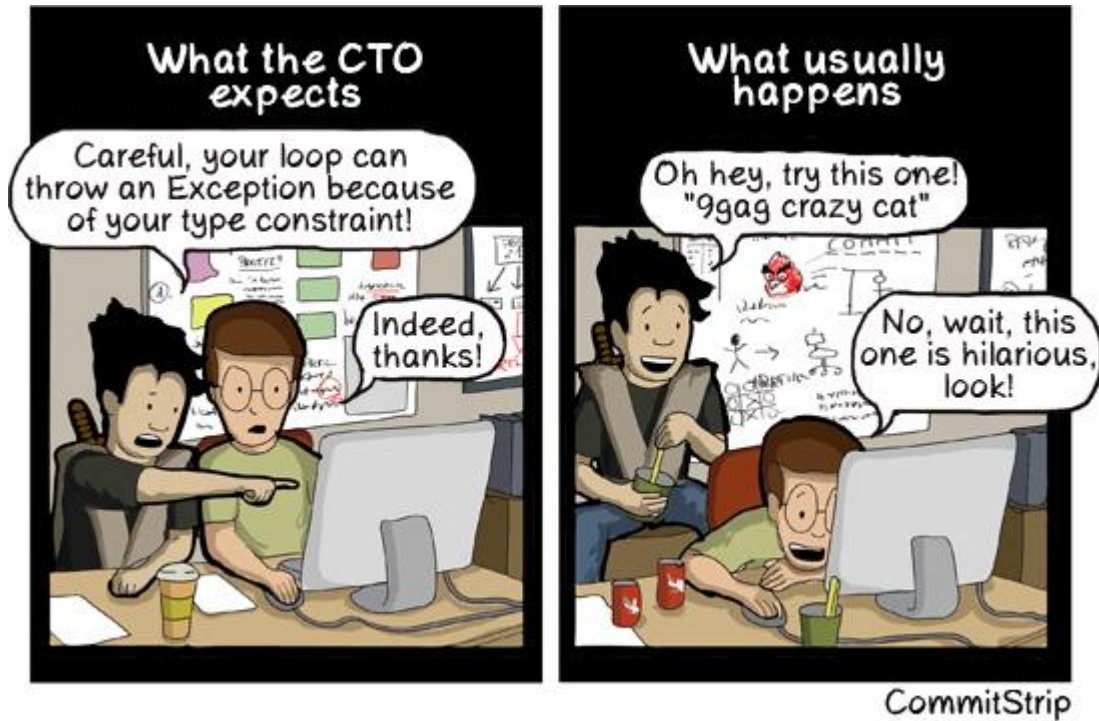
XP

- For synergy:
 - Pair programming
 - Have all production code written with two programmers actively at one machine
 - Prevents individual code ownership

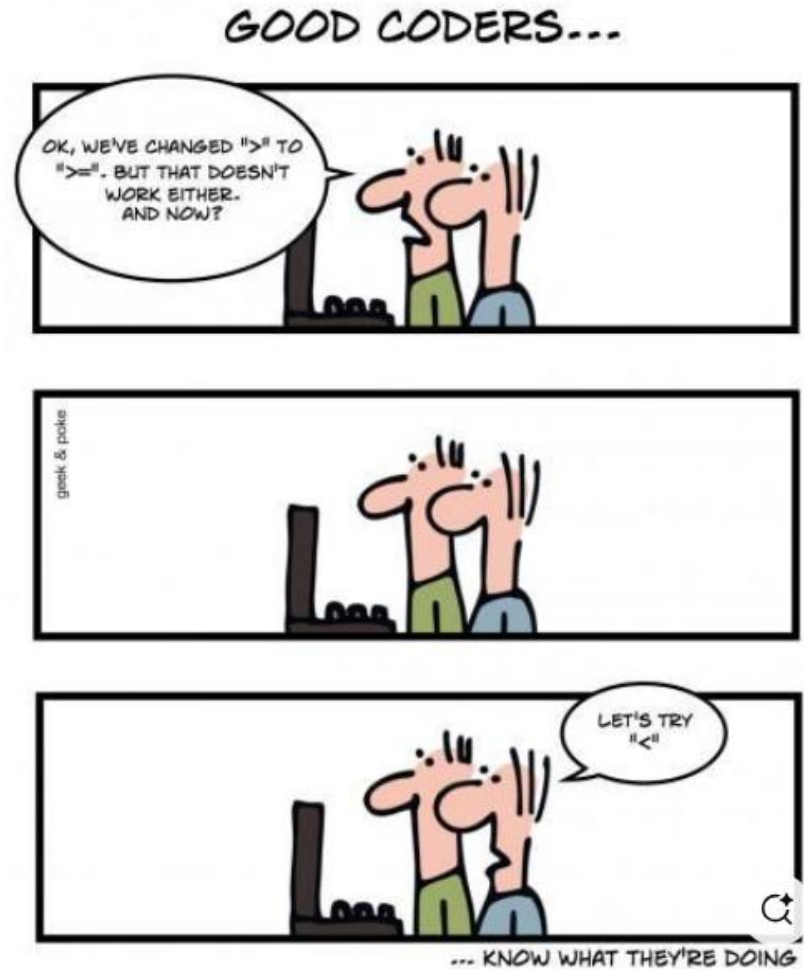
XP

- So why is it called “extreme”?
 - If short iterations are good, make them as short as possible
 - If simplicity is good, make the simplest thing that works
 - If design is good, do it all the time (refactoring)
 - If testing is good, write tests first, and do it all the time (test-driven development)
 - If code reviews are good, do it all the time (pair programming)

"Pair Programming"



<https://www.commitstrip.com/en/2012/08/14/pair-programming/>



© Geek & Poke

Discussion

- What are reasons for having programmers working in pairs?
- What are reasons they shouldn't?

Pair Programming

- Synergies:
 - More ideas
 - Complementary skills
 - Better consideration of alternative solutions
 - Learning
 - Expert/student apprenticeship
 - Continuous critique to learn new things

Pair Programming

- Synergies:
 - Pressure
 - They do not want to let each other down, or waste each other's time
 - Courage
 - They give each other confidence to do things they might avoid if alone

Pair Programming

- Synergies:
 - Reviews
 - Better able to reveal defects with more eyes looking at the code
 - Debugging
 - Bugs reveal themselves when one explains the misbehaving code to the other

Scrum

- One part of an agile development process
- Based on:
 - Feedback, roles, meetings, prioritization and planning
- Like classic engineering management, and is often used onsite in civil engineering

Scrum

- Roles:
 - Scrum master
 - Knows the process (Agile, XP, etc.)
 - Protects the team and helps the team follow Scrum
 - Product owner
 - Represents the customer
 - Team members
 - Write code

Scrum

- Meetings:
 - Many per iteration
 - Daily scrum
 - Once per iteration
 - Planning meeting
 - Review
 - Retrospective

Scrum

- Daily scrum:
 - AKA daily “standup”
 - Time limited
 - Everyone is standing, so they are more uncomfortable and want to finish soon
 - Each team member answers 3 questions:
 - What did you do?
 - What are you going to do?
 - What is blocking you?

Scrum

- Planning meeting:
 - First meeting of the iteration (only on first day)
 - Input: requirements and user stories
 - Output: choose appropriate stories to work on next
 - Estimate their cost in time
 - Prioritize them
 - Fit them into the time left for the iteration

Scrum

- Review:
 - Review work completed
 - Review work not completed
 - Demonstrate current system

Scrum

- Retrospective:
 - Review issues faced with quality and personnel
 - Try to improve the process
 - What went well?
 - What could be improved?
 - Stay calm

More Information

- Articles:
 - “A Rational Design Process: How and Why to Fake It”
 - D. L. Parnas and P. C. Clements
 - IEEE TSE, 12(2), 1986
 - “Software Development Worldwide: The State of the Practice”
 - M. Cusumano, A. MacCormack, C. F. Kemerer, and W. Crandall
 - IEEE Software, November/December 2003
 - “How Microsoft Builds Software”
 - M.A. Cusumano and R.W. Selby
 - Comm. ACM, 4(6), 1997

More Information

- Books:
 - Software Project Survival Guide
 - S. McConnell
 - Microsoft Press, 1998
 - The Build Master
 - V. Maraia
 - Addison-Wesley, 2005
 - Extreme Programming Explained
 - K. Beck
 - Addison-Wesley, 2004
 - Pair Programming Illuminated
 - L. Williams and R. Kessler
 - Addison-Wesley, 2002

More Information

- Books:
 - Software Project Survival Guide
 - S. McConnell
 - Microsoft Press, 1998
 - The Build Master
 - V. Maraia
 - Addison-Wesley, 2005
 - Extreme Programming Explained
 - K. Beck
 - Addison-Wesley, 2004
 - Pair Programming Illuminated
 - L. Williams and R. Kessler
 - Addison-Wesley, 2002