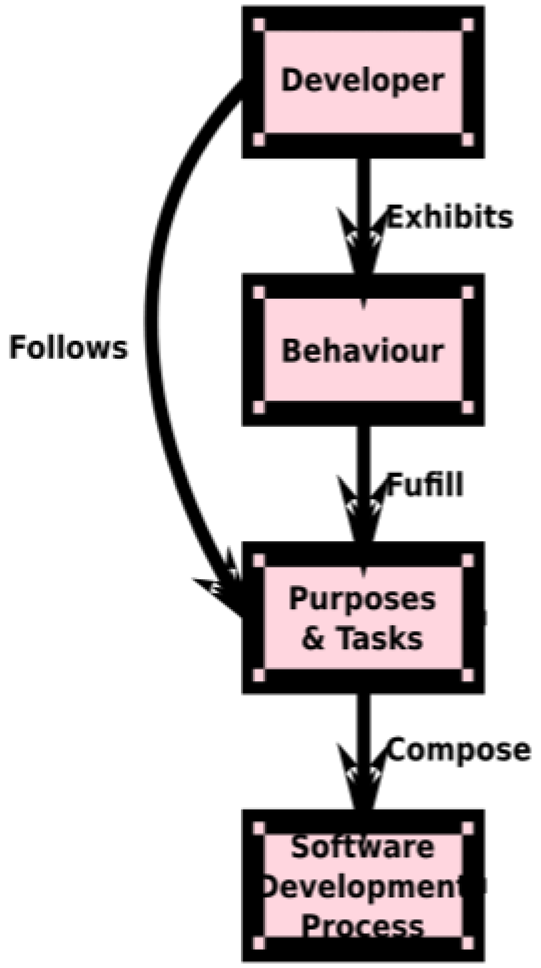
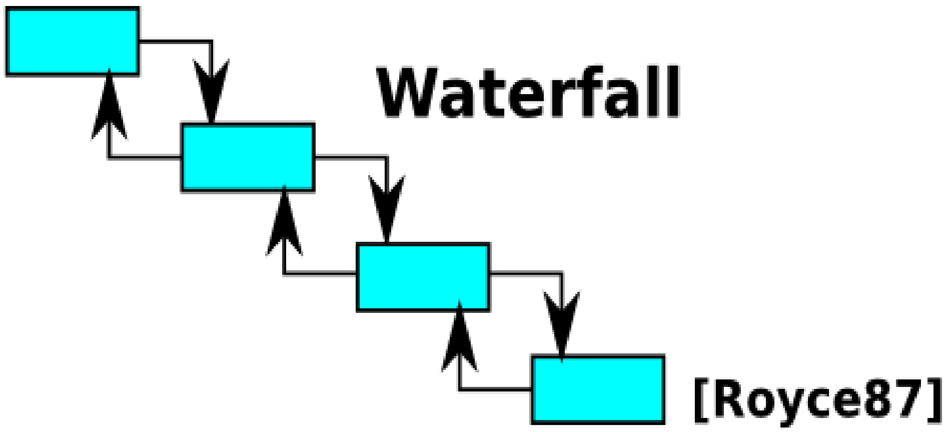
Software Process

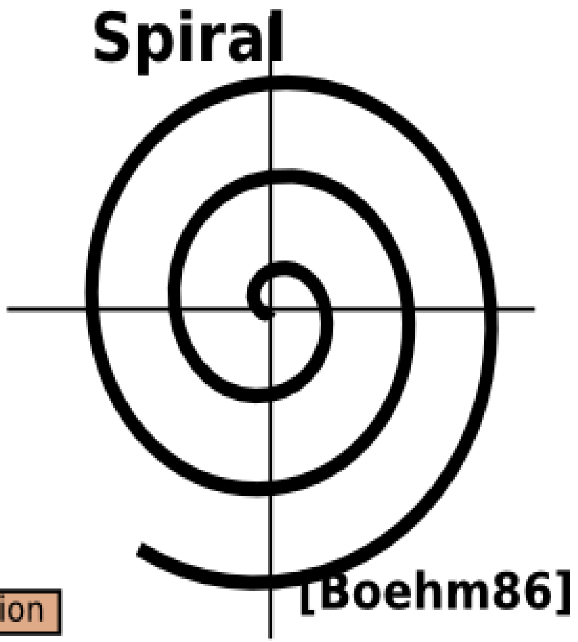
Dr. Hazel Campbell
Dr. Abram Hindle
Dr. Ken Wong
Department of Computing Science
University of Alberta

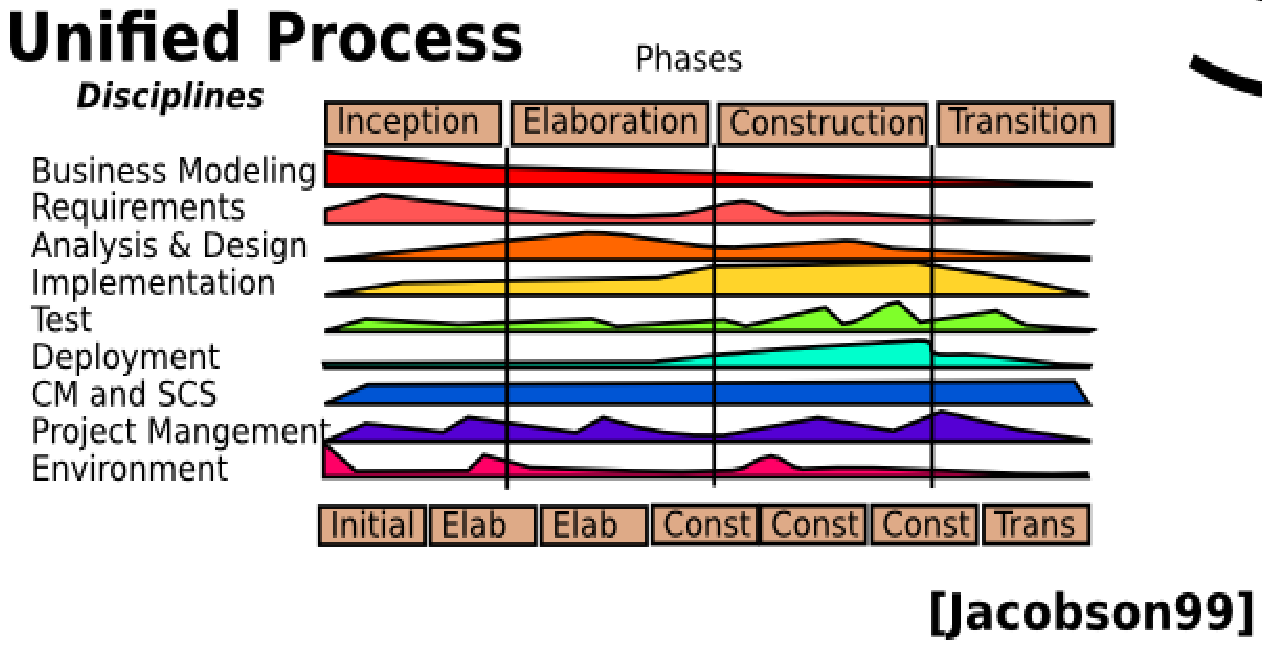
What makes a Process?



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

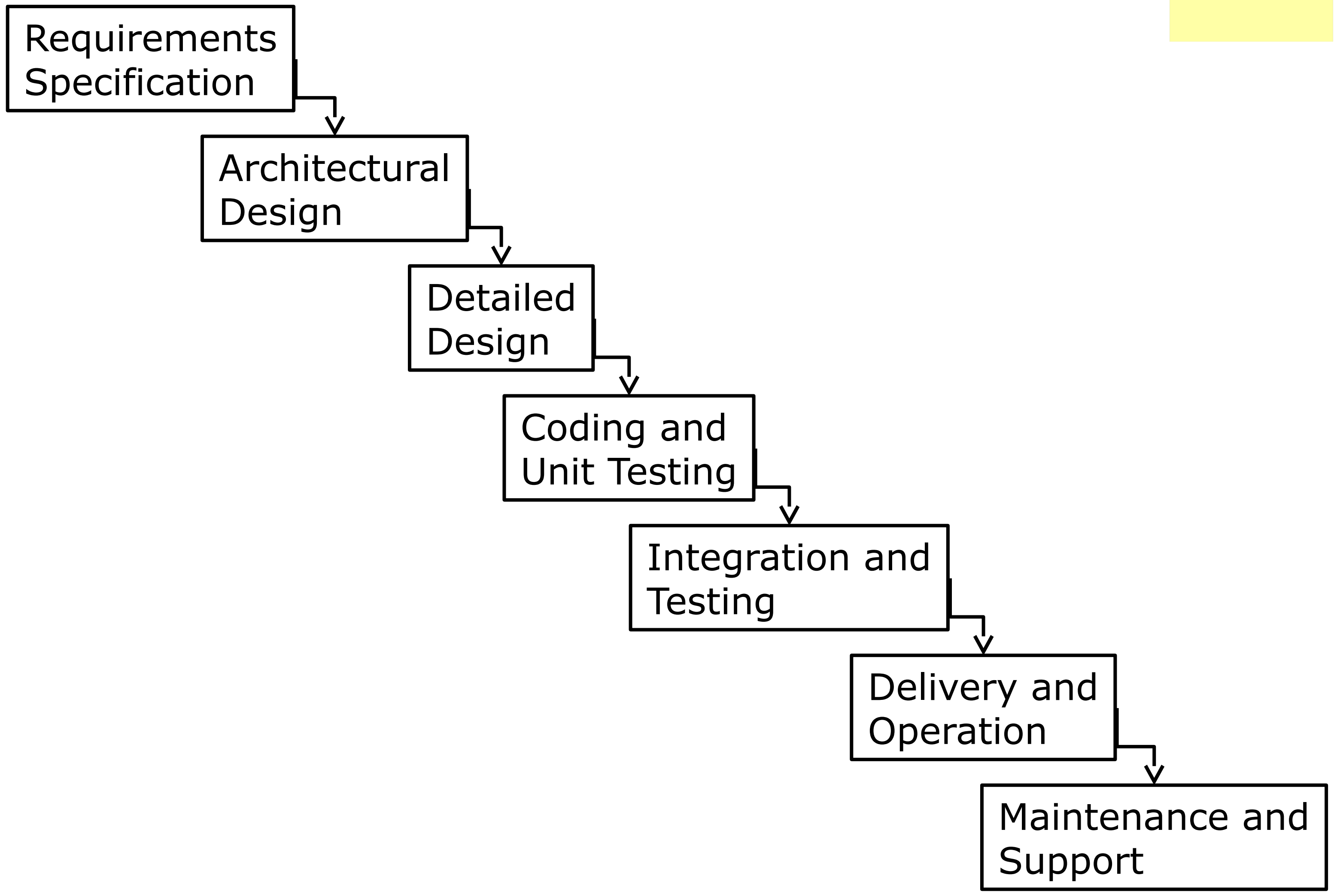
User Perspective

- User Experience (UX):
  - Satisfying
  - Motivating
  - Looks nice (aesthetically pleasing)
  - Enjoyable
  - Fun

Meeting Needs

- Verification
  - making sure you develop the system right
    - according to the requirements

Waterfall Lifecycle 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
  - Requirements need to be right (accurate) at the start
    - This is almost never the case
    - Could end up building the wrong system
    - Hard to predict all necessary requirements
    - Hard to react to changing requirements
- Waterfall doesn’t work
  - We need to be able to iterate!

Prototyping

- It’s hard to get the requirements right at the start…
- But we need validation…
  - making sure we develop the right system
  - Making sure we build what the customer really needs
- One solution: Prototyping!

Prototyping

- Iterative design
  - Cycling through several designs
  - Improve the product with each pass

Prototyping

- Types of prototyping:
  - Throwaway Prototyping
  - Incremental Prototyping
  - Evolutionary Prototyping
- These can be combined!

Throwaway Prototyping

- Process
  - Build and test prototype
  - Learn about:
    - What’s needed for the real product
    - What works
    - What does not work
  - Throw away the prototype
  - Then develop the real product

Throwaway Prototyping

- Pros
  - more communication between users and developers
  - functionality is introduced earlier, which is good for morale

Throwaway Prototyping

- Cons
  - The throwaway prototype must be built very quickly
  - 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”
    - Example: “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

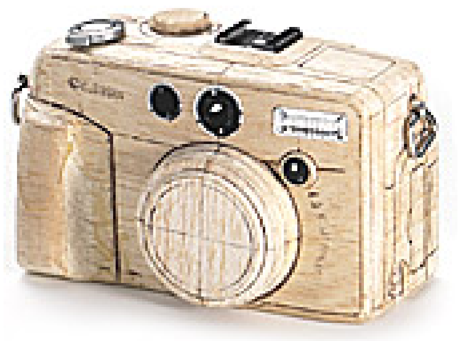
- Process
  - Each feature is refined or “evolved” over time
- Example: Text Editor
  - Prototype #1: Keyboard Cut and Paste
  - Prototype #2: Touchscreen Cut and Paste
  - Prototype #3: Cut and Paste works with Undo

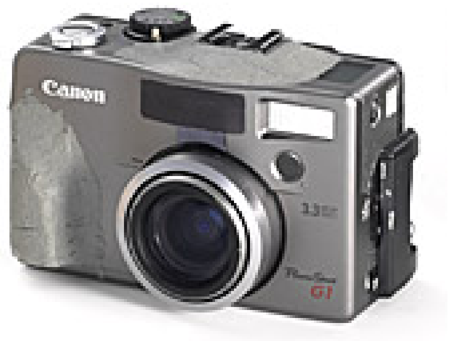
Other Kinds of Prototypes

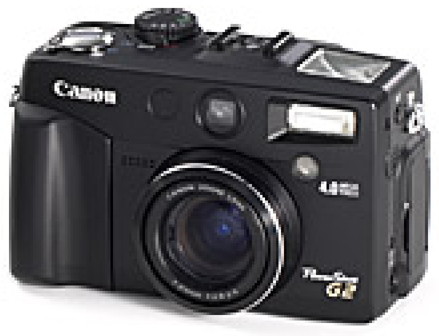
- User Interface Sketches
  - Hand Drawn
  - or using a drawing Tool
    - Figma, Balsamiq, etc.
- Storyboards
  - Graphical depiction of the user interface
  - Like a comic strip, but only draw the UI

Other Kinds of Prototypes

- Physical Mockups

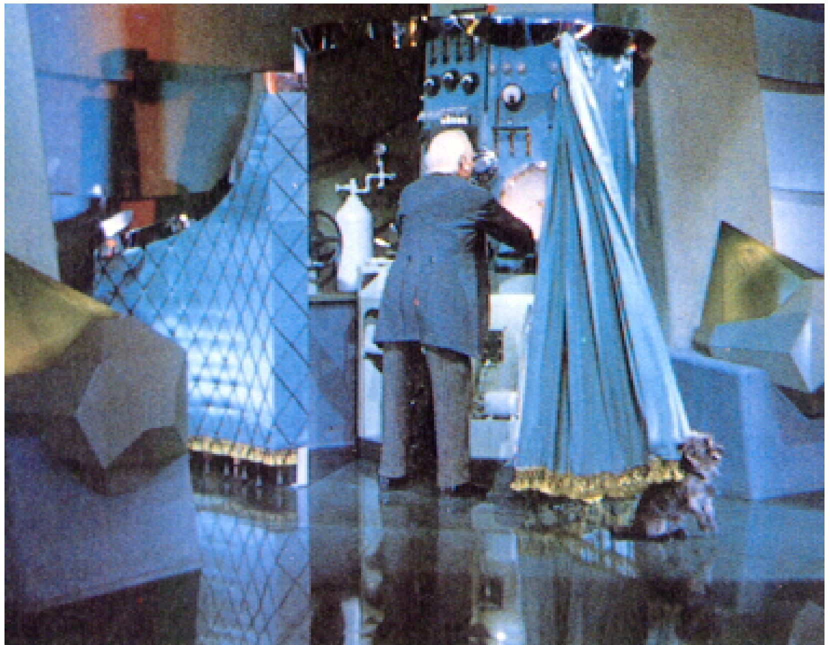

Balsa wood mock-up


Partial clay mock-up


Precision mock-up

Other Kinds of Prototypes

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



© MGM

Agile Practices

- Created with the release of the
- “Agile Manifesto”
  - http://agilemanifesto.org/

4 Agile Values

- “Individuals and Interactions”
- “Working Software”
- “Customer Collaboration”
- “Responding to Change”

4 Agile Values

- "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

4 Agile Values

- “Working software”:
  - the main measure of progress
  - continuous, frequent delivery of value

4 Agile Values

- “Customer collaboration”:
  - customers and developers work together
  - satisfy customer early

4 Agile Values

- “Responding to change”:
  - welcome changing requirements, even late
  - technical excellence and good design
  - simplicity—art of maximizing work not done

Interlude: Prioritizing Stories (Features)

- High priority
  - **Must** be done
  - Do it first
  - Risk level:
    - Will cause big problems if we don't do it (first)
    - Will cause big problems if it breaks

Interlude: Prioritizing Stories (Features)

- Medium priority
  - **Should** be done
  - Do it second
  - Risk level:
    - Will cause some problems if we don’t do it
      - (before other user stories)
    - Will cause some problems if it breaks

Interlude: Prioritizing Stories (Features)

- Low priority
  - **Could** be done
  - Do it third
  - Risk level:

      Only minor problems if we don’t do it
      - (before other user stories)
    - Only minor problems if it breaks

Interlude: Prioritizing Stories (Features)

- No priority
  - We'd like it but we **won't** get it
  - Do it last

  - Risk level:
    - No problems if we don't do it
      - (before other user stories)

Interlude: Estimating Cost

- Agile “flying fingers” method – wisdom of the crowd!
  1) Read a user story, discuss it if necessary
  2) Then, everyone puts their hand behind their back, out of sight, holding up the number of fingers for the user story
  3) Someone counts: one… two… three…
  4) All the fingers come flying out at once!
  5) If the deviation is small – choose a mean and write it down. Move to the next story.
  6) If there is substantial disagreement – discuss and repeat!

eXtreme Programming (XP)

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

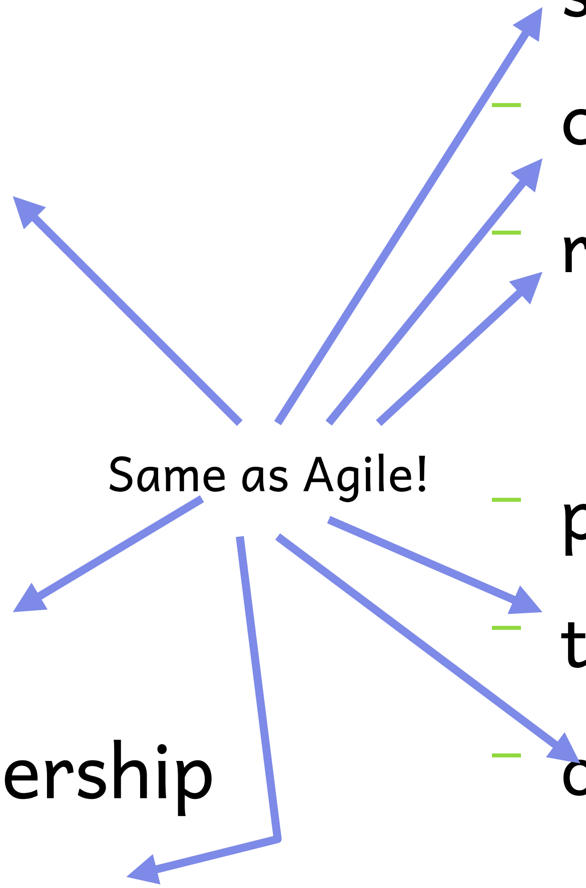
XP

- Philosophy:
  - communication
  - feedback
  - simplicity
  - programmer friendly
  - code-centric
  - for small teams (up to about 20)
  - 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 h 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 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 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 really short
- 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

- 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
    - planning
  - like classic engineering management, and is often used onsite in civil engineering

Scrum

- Roles:
  - Scrum master
    - knows the process (agile, xp...)
    - protects the team and helps the team follow Scrum
  - product owner
    - represents the customer
  - team members
    - write the code

Scrum

- Meetings:
  - daily scrum (1 per **day**)
  - planning meeting (1 per iteration)
  - review meeting (1 per iteration)
  - retrospective meeting (1 per iteration)

Scrum

- Daily scrum also known as 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 stories to work on next
    - estimate their cost in time
    - prioritize them
    - fit them into the time left for the iteration

Scrum

- Review Meeting
  - review work completed
  - review work not completed
  - demonstrate current system

Scrum

- Retrospective Meeting
  - 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

More Information

Articles:

“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

More Information

Books:

Extreme Programming Explained

- K. Beck
- Addison-Wesley, 2004

Pair Programming Illuminated

- L. Williams and R. Kessler
- Addison-Wesley, 2002