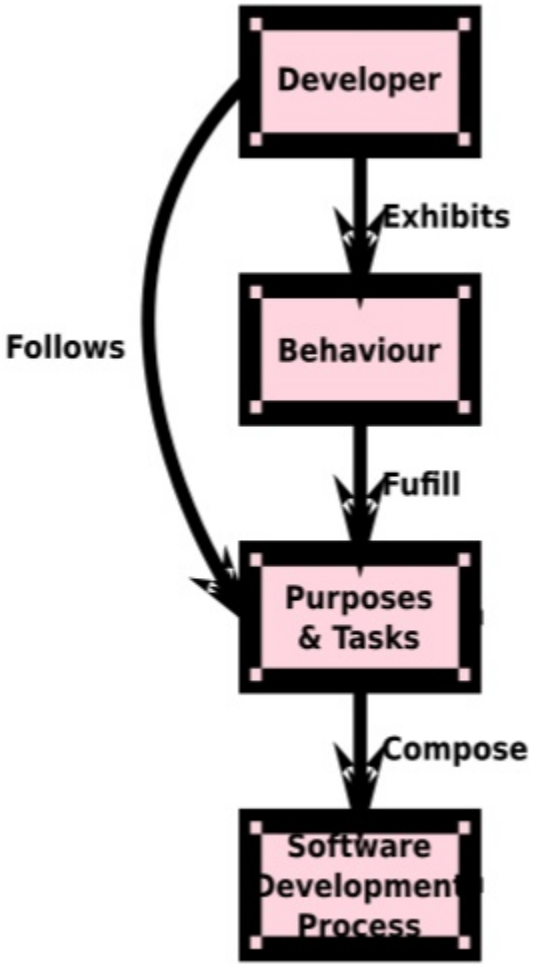
**Abram Hindle**
Department of Computing Science
University of Alberta

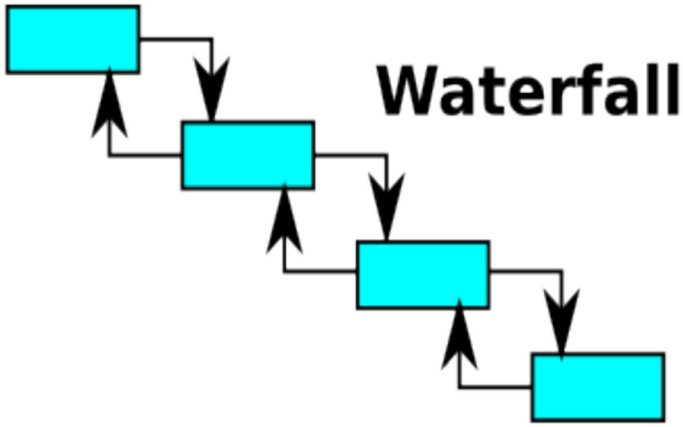
# Software Process

Version 1.1

# What makes a Process?



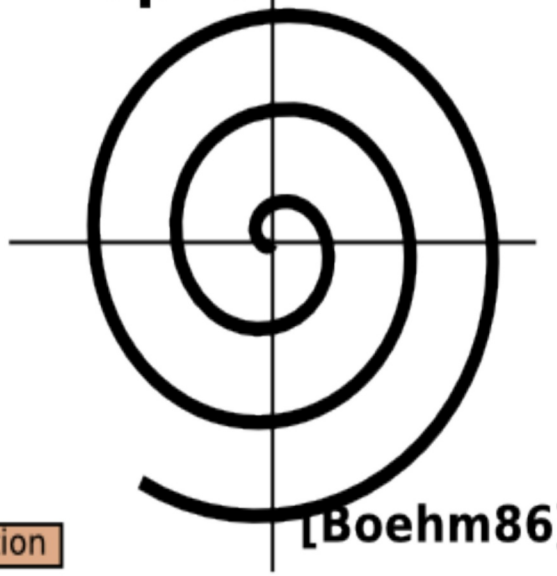
# Software Development Processes



Waterfall

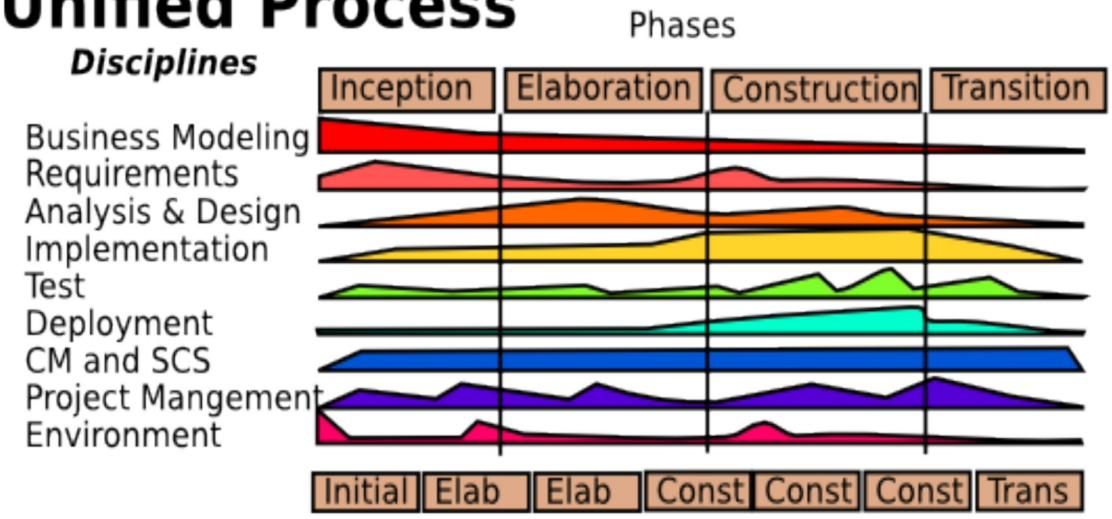
[Royce87]

Spiral



[Boehm86]

## Unified Process




[Jacobson99]

* CMM
* SDLC

# Developer Perspective

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

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)

# Discussion

Question:

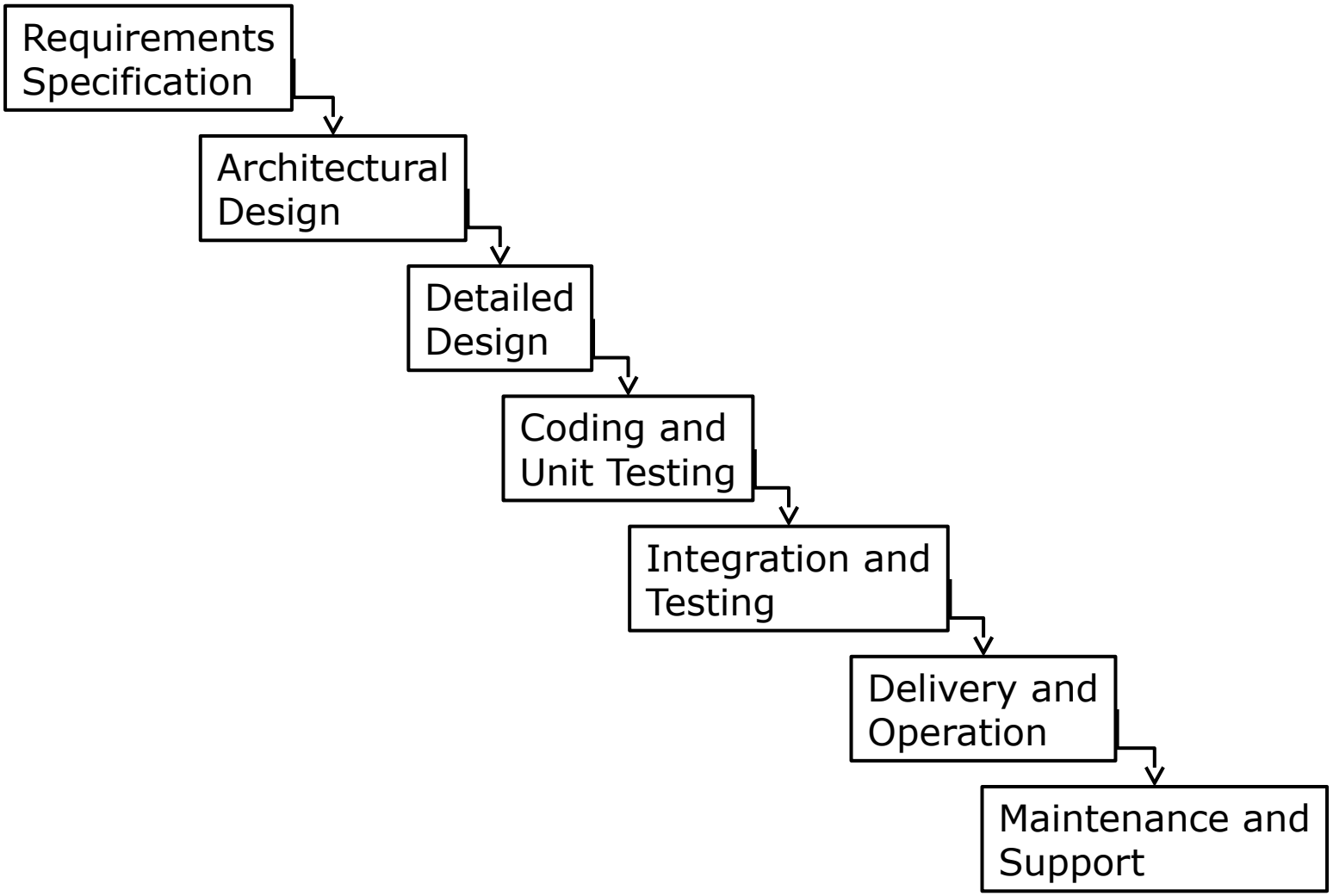
What are some major activities in developing software?

Question:

Is there an effective order on these activities?

- Waterfall

# Waterfall Lifecycle Model



# Discussion

Question:

What are some pros and cons of the waterfall model?

# Waterfall

Pros:

- easily understood
- enforces discipline
- verification at every phase
- documentation

# 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

# Waterfall

Cons:

dependence on requirements being “right”

- could end up building the wrong system

requirements must all be known up front

- but cannot always foresee all the requirements

Summary

need to be able to iterate

- Prototyping

# Meeting Needs

Validation

making sure you develop the *right system*
(i.e., what the customer really wanted)

# 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

“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 — command key cut/paste

prototype 2 — undoable cut/paste

prototype 3 — drag and drop cut/paste

# Other Kinds of Prototypes

User interface sketches

- hand drawn or using drawing tool

Storyboards

- graphical depiction of user interface
- like a comic strip

# 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

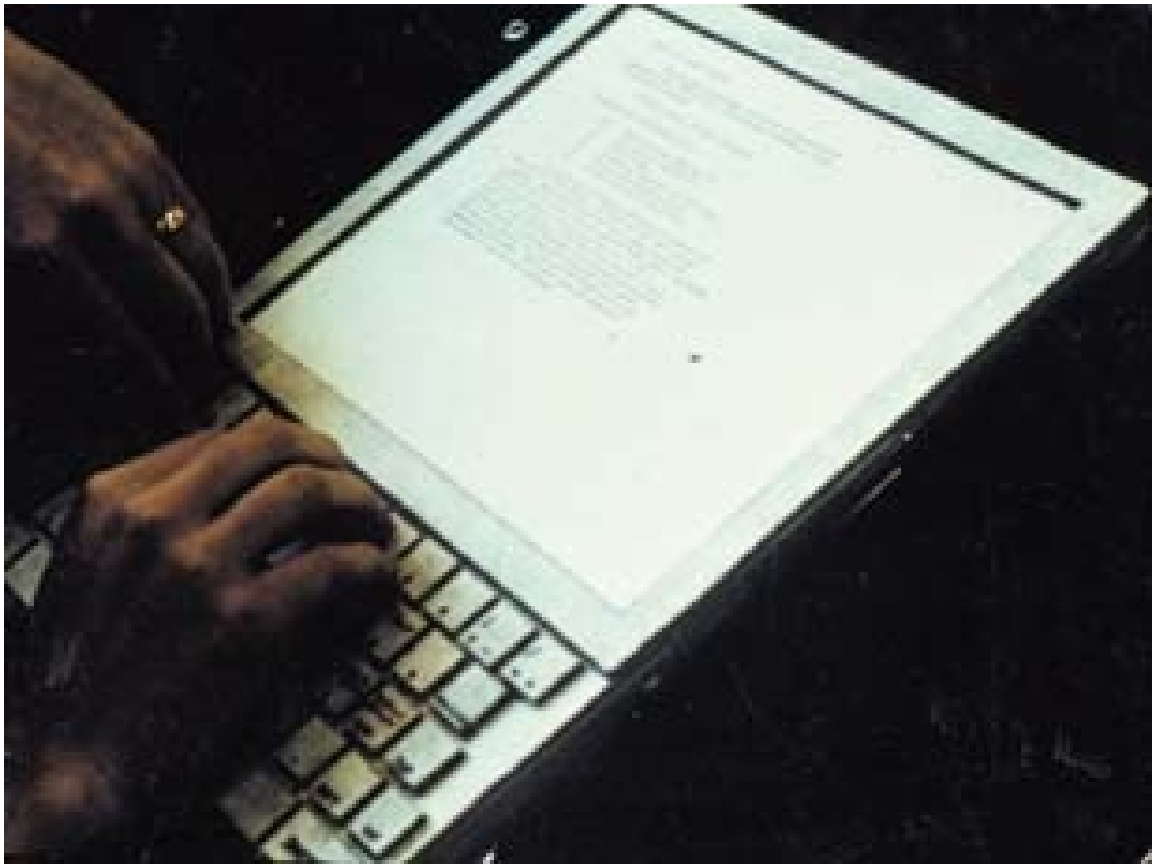
Physical mockups:

e.g., made out of wood, clay, or foam

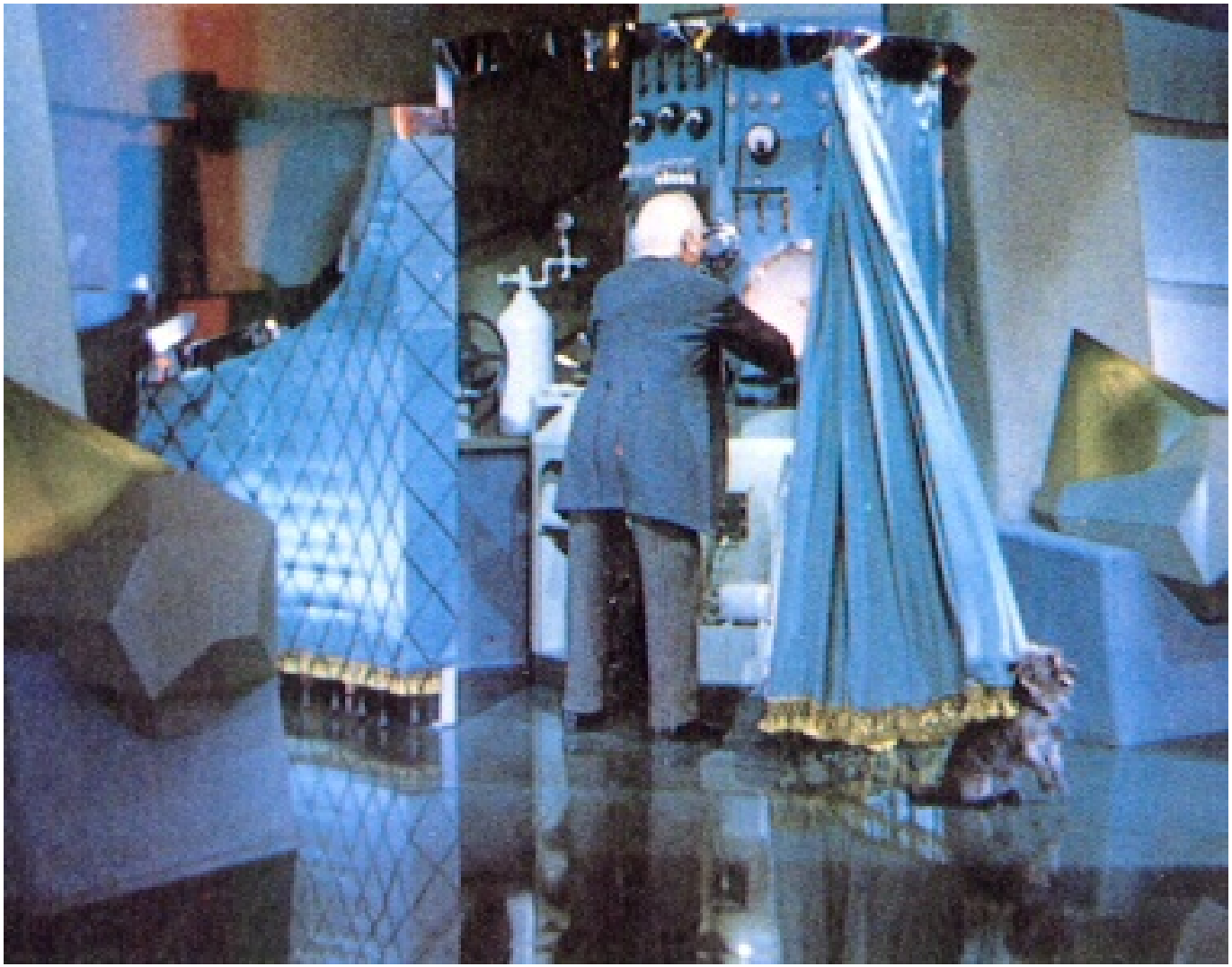
Balsa wood mock-up


Partial clay mock-up


Precision mock-up



© Alan Kay



# 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”

- Staged Delivery

# Staged Delivery

Developers:

deliver the system in a series of working releases or builds

Users:

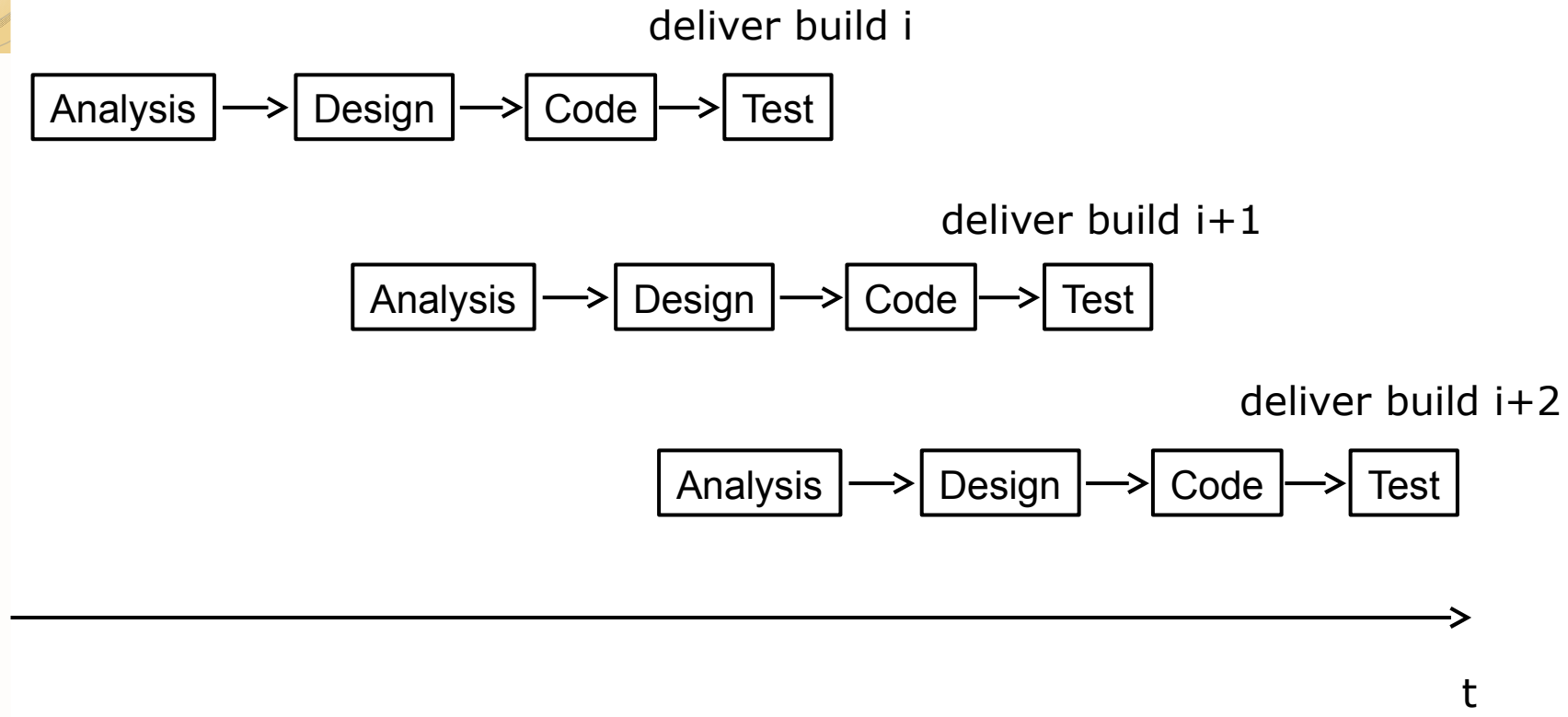
use some functionality while the rest continues to be developed

Possible parallelism:

production and development systems

staggered development streams

# Staggered Builds



# Staged Delivery

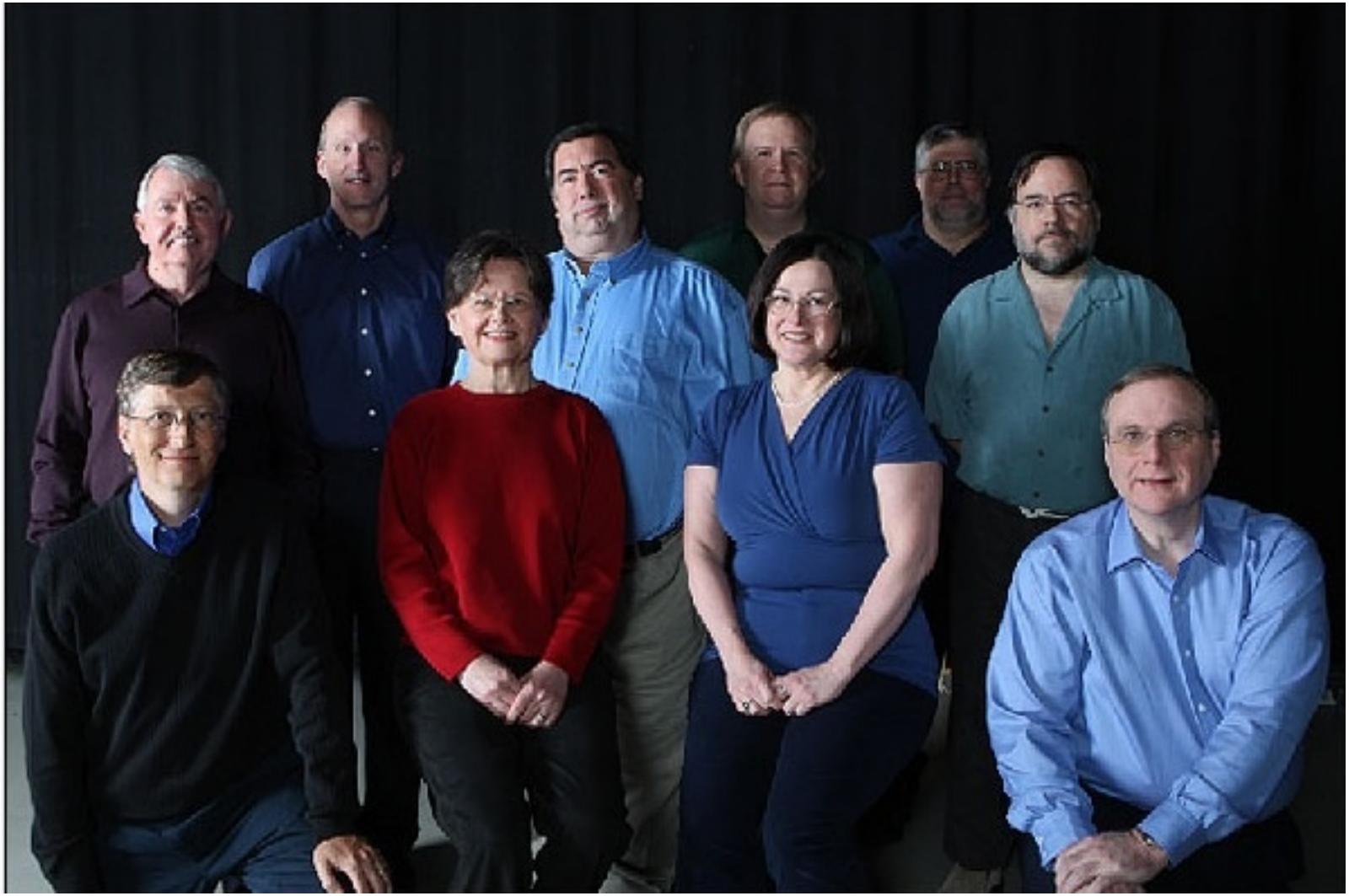
Pros:

- 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



# Microsoft Daily Build

Process:

- software product is built every day
- build cycle becomes the heartbeat of the project; everyone knows the status
- built system must be runnable for overnight testing

# Microsoft Daily Build

Testing:

if the build breaks (not runnable nor testable), the whole process is stopped until the problem is found

failures detected during testing are available and broadcast next morning

huge incentive not to break the build

# Continuous Integration

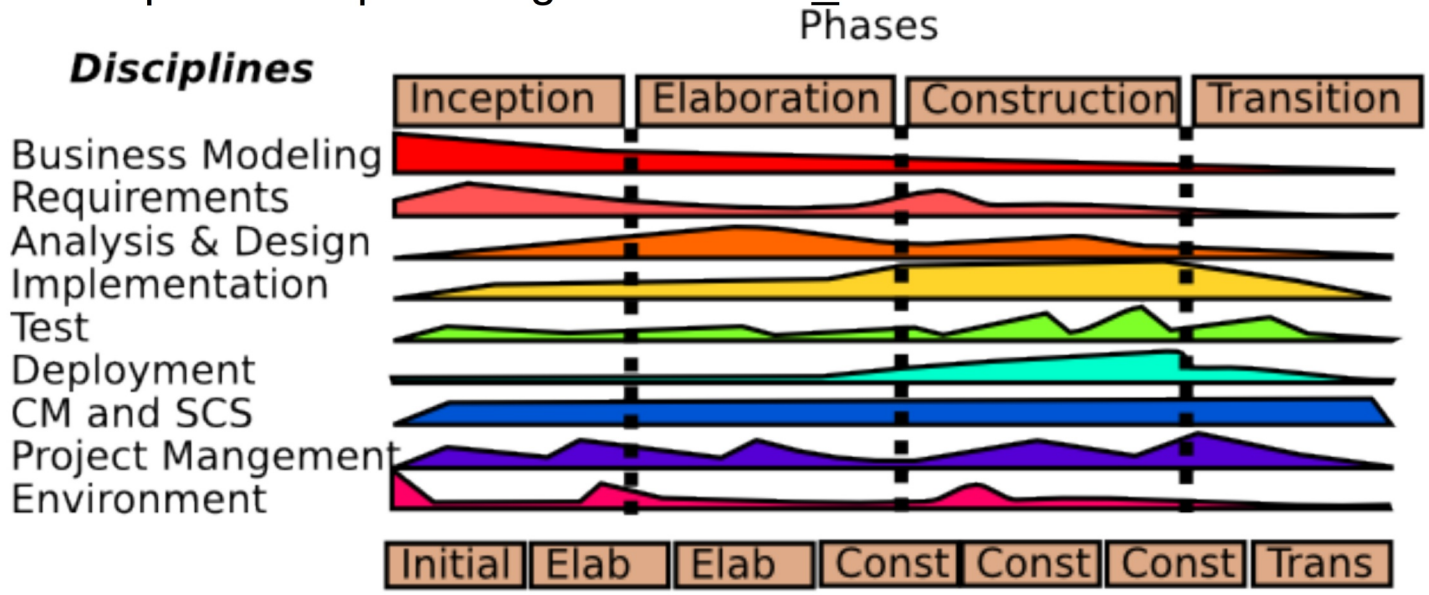
- Take the daily build
  - Combine it with testing
  - Combine it with building
  - Maybe combine it with deployment
  - Do it continuously (repeatedly)
  - Do it as much as a possible (per commit)
- Use tools such as:
  - Hudson
  - Jenkins
  - Travis-CI
  - Microsoft Team Foundation Server
  - Apache Continium, Apache Gump
  - Tinberbox

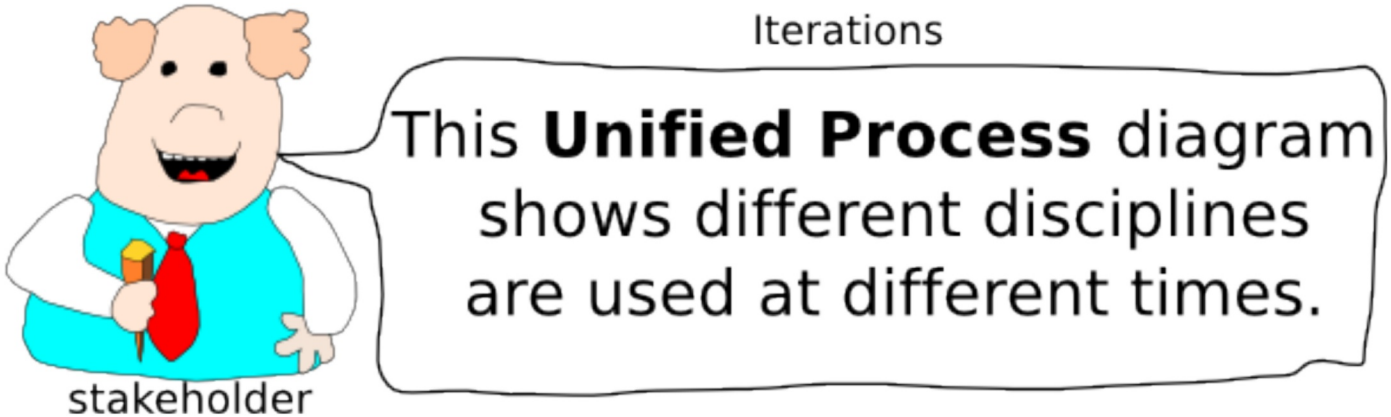
- Unified Process

# Unified Process

Link:

http://en.wikipedia.org/wiki/Unified_Process



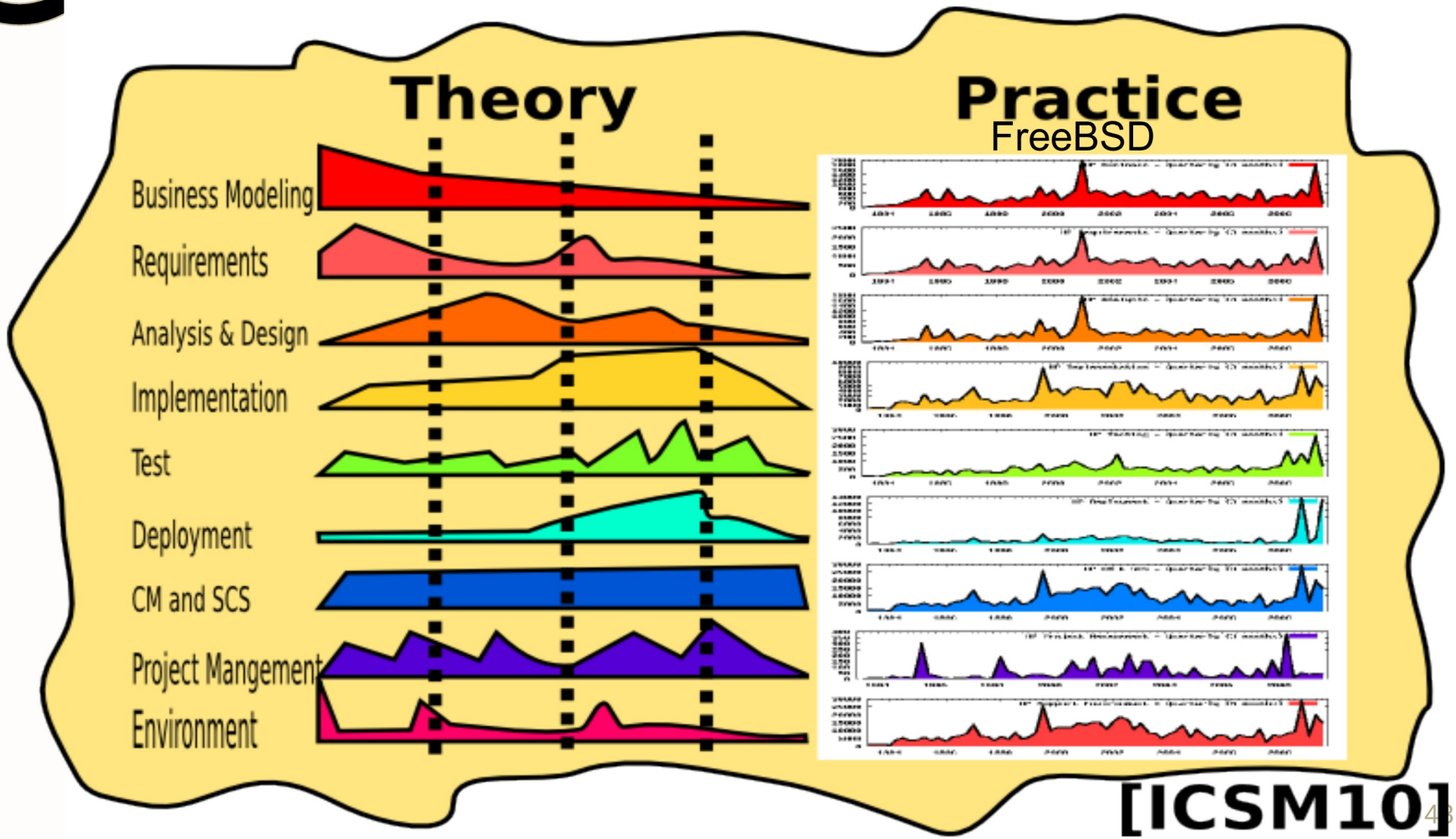



# Unified Process

* Iterative
* Incremental
* Customizable
* Phases
    * Inception: Risks and Business Cases and Use Cases
    * Elaboration: use case diagrams and class diagrams
    * Construction Phase: implementation in iterations
    * Transition: Deployment

# Unified Process

Link:
http://en.wikipedia.org/wiki/Unified_Process




[ICSM10]

- Agile Practices

# “Agile Manifesto”

Link:

http://agilemanifesto.org/

# 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 late

technical excellence and good design

simplicity—art of maximizing work not done

# eXtreme Programming (XP)

Link:

http://www.extremeprogramming.org/

# XP

Philosophy:

- communication
- feedback
- simplicity

- programmer friendly
- code-centric
- for small teams (up to about 20)

- requires courage

# 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 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 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

# “Pair Programming”

http://www.dilbert.com/strips/comic/2003-01-09/

# Discussion

Question:

Why should programmers work in pairs?

# 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

# 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)

# Scrum

- Agile Process
- Doesn't prescribe many development methods
- Based around
  - Feedback
  - Roles
  - Meetings
  - Prioritization and Planning
- Scrum is like classic engineering management processes and is often used onsite in civil engineering.

# Scrum Roles

- Scrum Master
  - Process Master, protects the team and helps the team follow scrum
- Product Owner
  - Represents the customer
- Team members

# Scrum Meetings

- Planning Meeting (1 per iteration)
- Daily Scrum ( many per iteration)
- Review (1 per iteration)
- Retrospective (1 per iteration)

# Scrum Meetings

- Planning Meeting
  - First meeting of the iteration (1 day)
  - Take requirements and user stories and:
    - Choose appropriate stories to work on next
    - Estimate their cost in time
    - Prioritize them
    - Fit them into the time left for the iteration.

# Scrum Meetings

- Daily Scrum
  - Also the daily standup
  - Everyone stands up so that they are uncomfortable and want to finish soon
  - Time limited
  - Every team member answers 3 questions:
    - What did you do?
    - What are you going to do?
    - What is blocking you?

# Scrum Meetings

- Retrospective
  - Review issues faced with quality and personel
  - Try to improve the process
  - What went well?
  - What could be improved?
  - Stay Calm
- Review
  - Review work completed
  - Review work not completed
  - Demonstrate current system

# Some Scrum in the lab

- I define my user stories in a text file.
- I act as the product owner, and tell the team what I want to see.
- The team decides what to work on next.
- Every day I ask my research assistants:
  - What did you do since last time?
  - What are you going to do?
  - What do you need from me?
- We don't explicitly prioritize
- We don't explicitly plan
- We don't have multiple iterations
  - Why not? Because we are experimenting and cannot plan more than a week ahead.

# 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