CASE WESTERN RESERVE
UNIVERSITY

Software Process Models

CSDS 393/493: Software Engineering

Spring 2025

Agenda

- Discuss project team forming
- Software process
  - Process models
  - Advantages and disadvantages
  - Improvement to the process models

Team Expectations

Team size: 4 members

- Meet initially and then regularly
- Review team policy
- Divide work and integrate
- Establish a process
- Set and document clear responsibilities and expectations

Tasks

- Project **teams due**: January 21
  - Spreadsheet will be available soon

- Project **proposal due**: January 28
  - One page document with project idea and possible features (informal)
  - Graded as Assignment-1

How to Develop Software?

1. Discuss the software that needs to be written
2. Write some code
3. Test the code to identify the defects
4. Debug to find causes of defects
5. Fix the defects
6. If not done, return to step 1

The Software Process

"The set of activities and associated results that produce a software product"

Four fundamental activities are common to all software processes

- Software specification
- Software development
- Software validation
- Software evolution

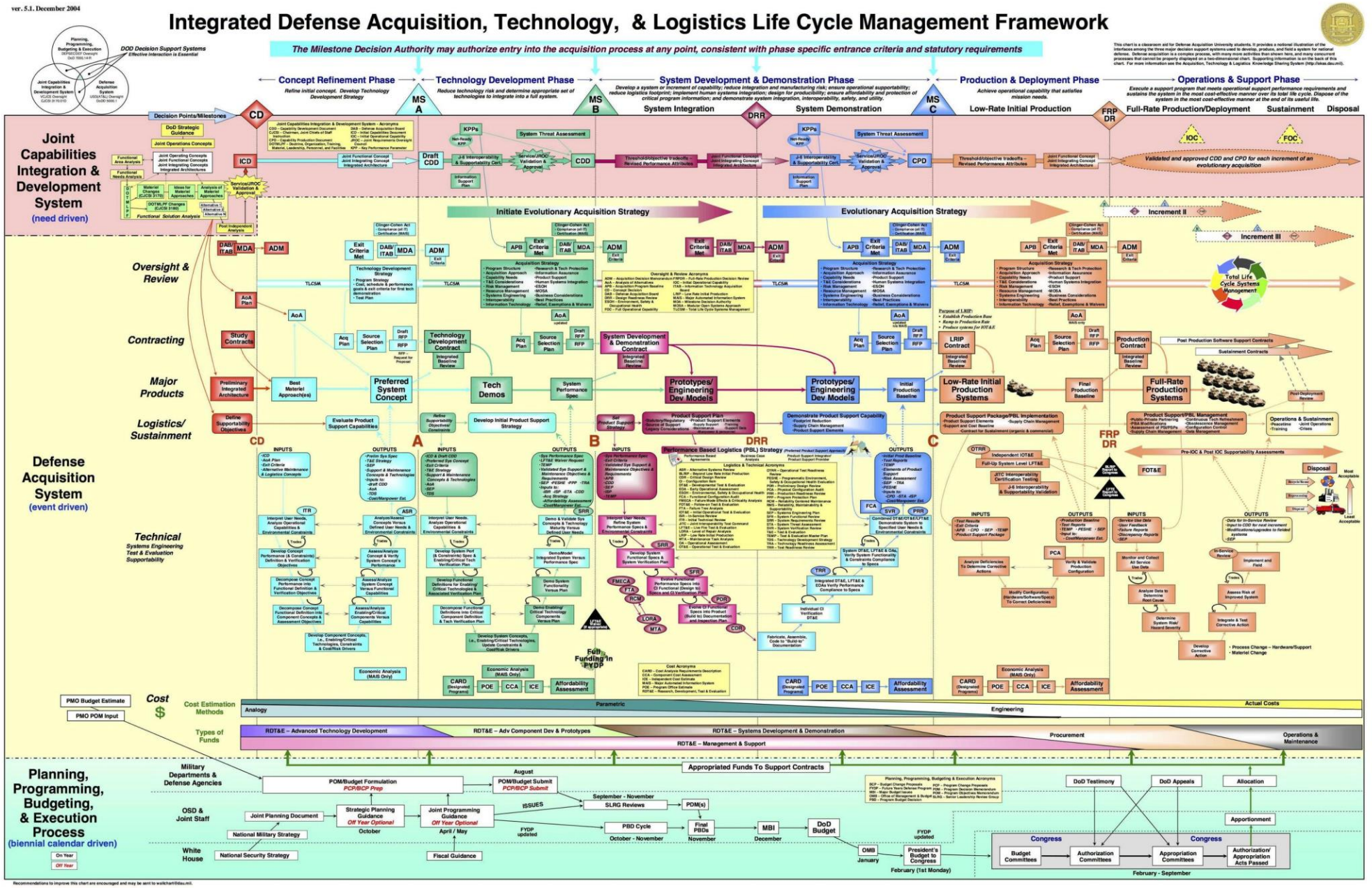
Software Process Descriptions

- Process descriptions may also include
  - Products: outcomes of a process activity
  - Roles: responsibilities of the people involved in the process
  - Pre- and post-conditions: statements that are true before and after a process activity

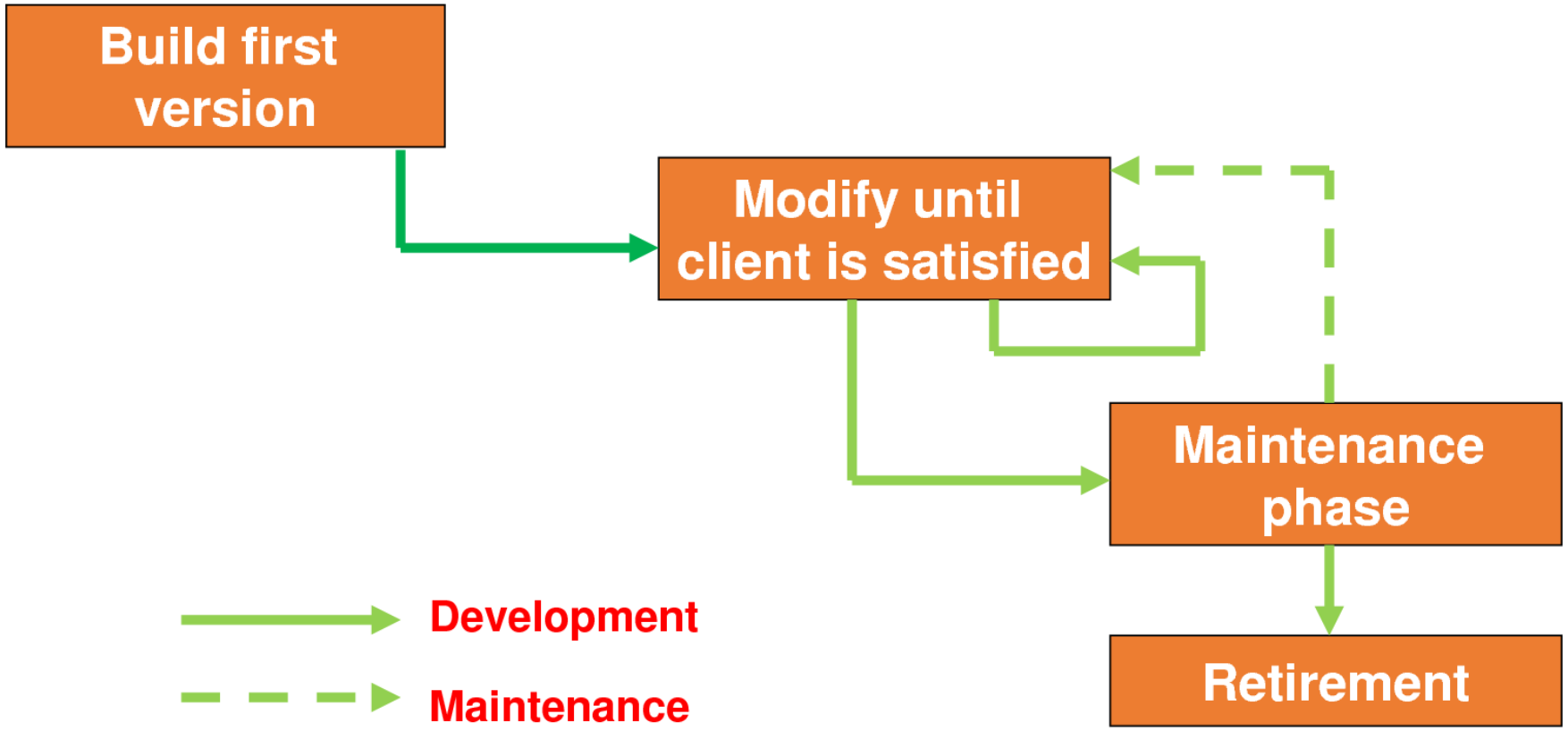
Software Process Model

Sometimes called a Software Development Life Cycle or **SDLC** model

- Graphical models of the software development process
- Characterize workflow between phases
- Have descriptive and prescriptive uses

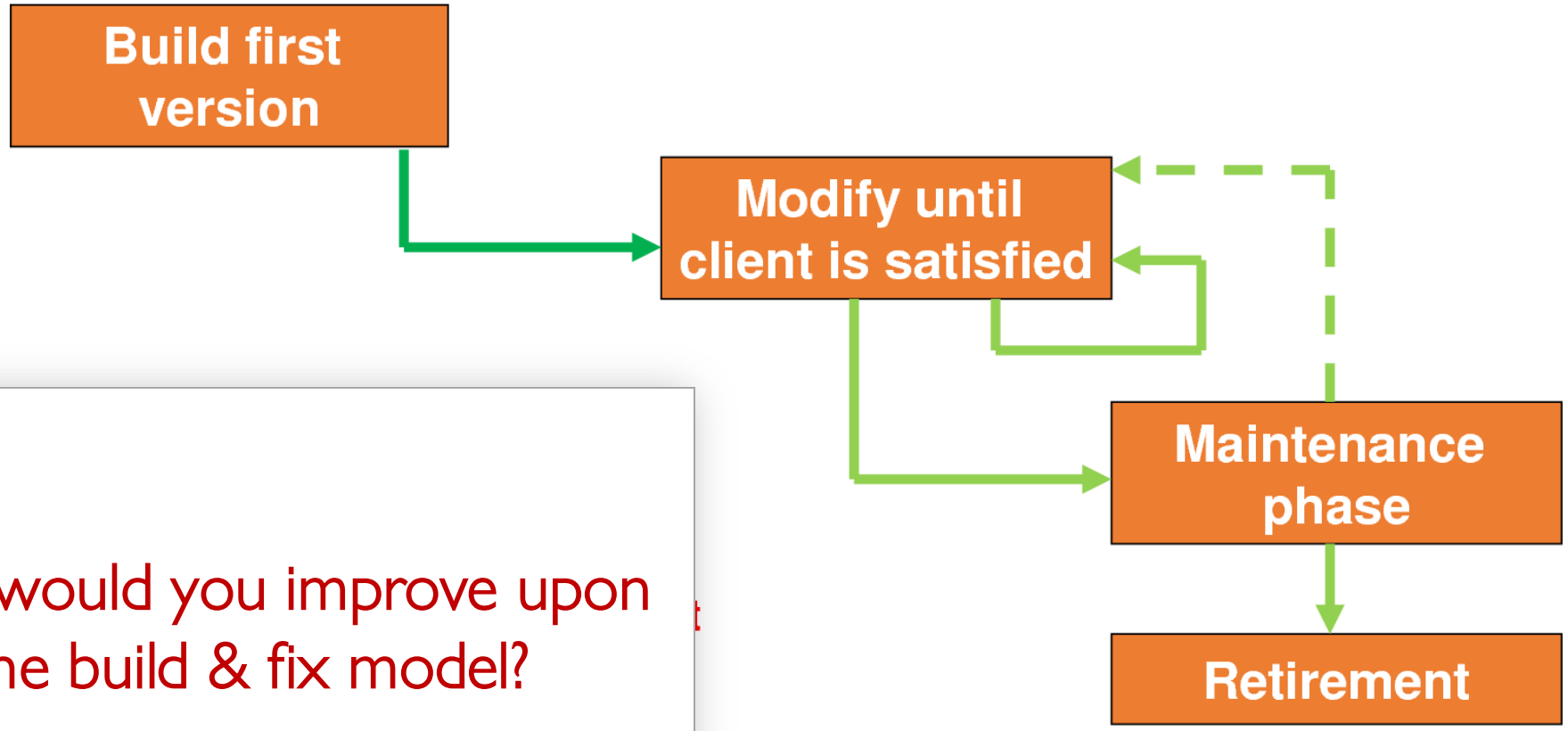

ver. 5.1. December 2004
Integrated Defense Acquisition, Technology, & Logistics Life Cycle Management Framework
DOD Decision Support Systems Effective Interaction is Essential
The Milestone Decision Authority may authorize entry into the acquisition process at any point, consistent with phase specific entrance criteria and statutory requirements
Concept Refinement Phase
Technology Development Phase
System Development & Demonstration Phase
Production & Deployment Phase
Operations & Support Phase
System Integration
System Demonstration
Low-Rate Initial Production
Full-Rate Production/Deployment
Sustainment
Disposal
Joint Capabilities Integration & Development System
(need driven)
Defense Acquisition System
(event driven)
Planning, Programming, Budgeting, & Execution Process
(biennial calendar driven)
Oversight & Review
Contracting
Major Products
Logistics/ Sustainment
Technical
Systems Engineering
Test & Evaluation
Supportability
Initiate Evolutionary Acquisition Strategy
Evolutionary Acquisition Strategy
Validated and approved CDD and CPD for each increment of an evolutionary acquisition
Increment II
Increment III
Cost
Cost Estimation Methods
Types of Funds
Appropriated Funds To Support Contracts
Military Departments & Defense Agencies
OSD & Joint Staff
White House
Congress
Congress
February - September

Anti-Model: Build & Fix



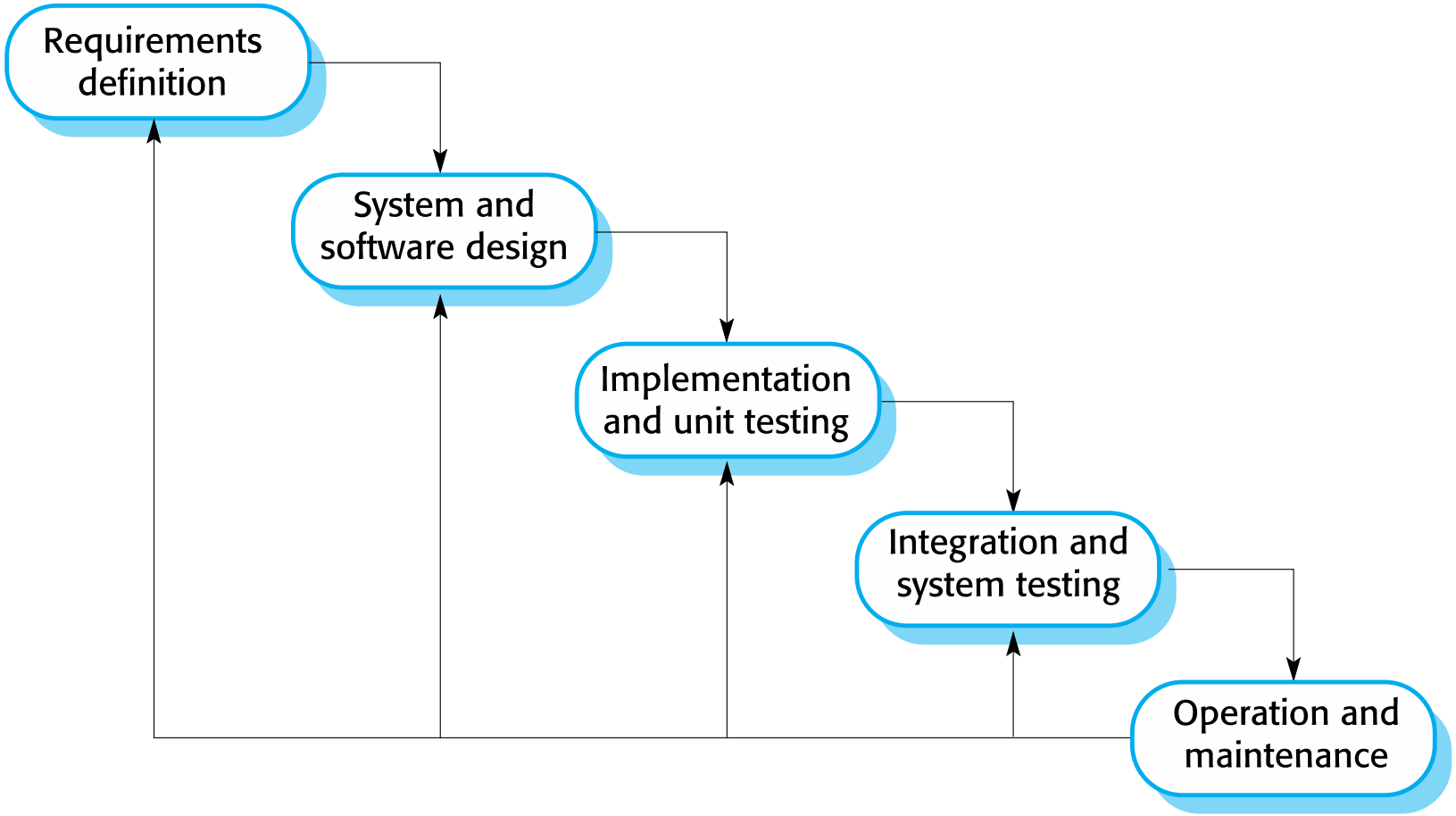
From Object-Oriented and Classical Software Engineering by Steven Schach

Anti-Model: Build & Fix



How would you improve upon the build & fix model?

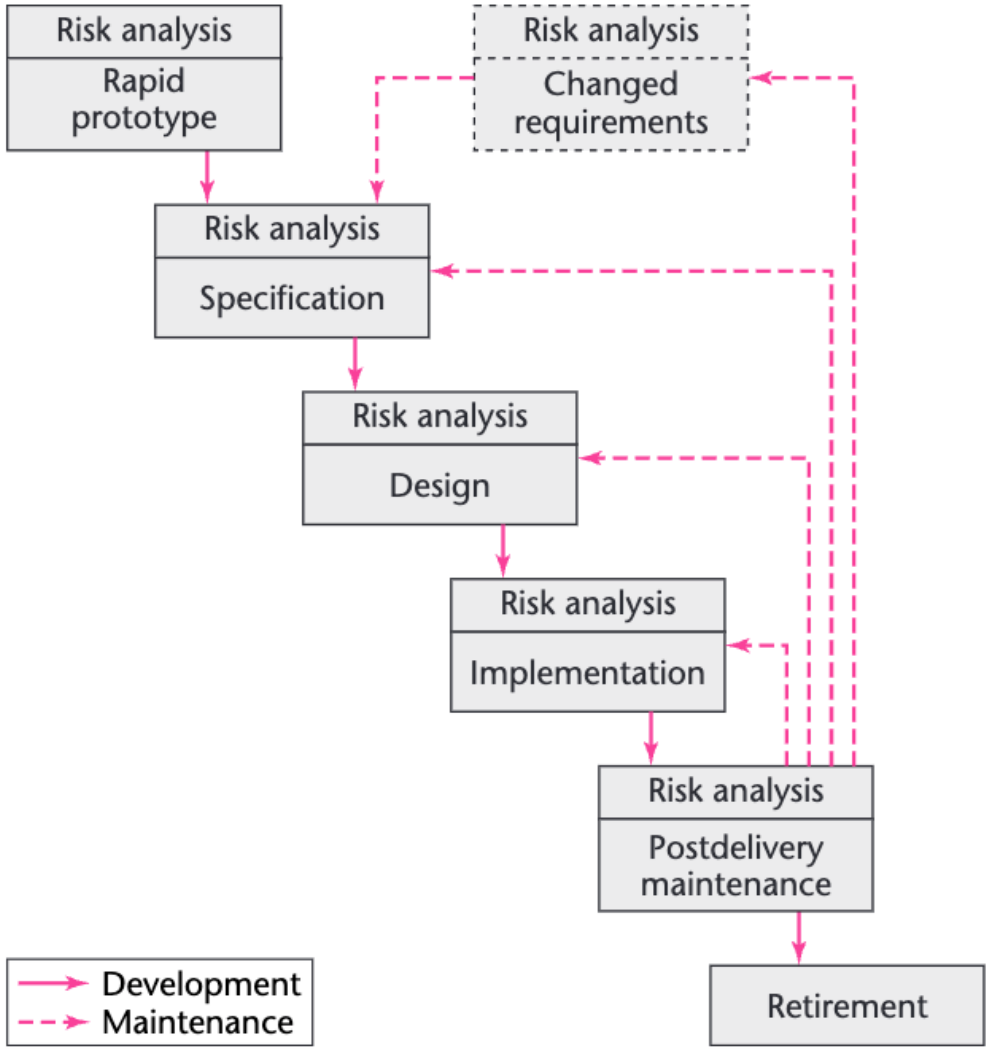
Waterfall Model



Basic Waterfall Model is Naive

- WFM is also thought as an anti-model in many cases
- It's often necessary to revisit earlier phases when problems are found with their products
- Thus, it's necessary to augment the basic model to reflect iteration and feedback

Waterfall Model Augmented with Iteration



Advantages of Waterfall Model

- Disciplined approach to development
- Careful analysis and documentation before coding can prevent costly problems later
- Documentation produced facilitates maintenance and training

Do you see the problem with the waterfall model?

Drawbacks of Waterfall Model

- It is difficult to convey dynamic appearance and behavior in a document
- Customers often know what they want only when they see it
- Requirements often change for other reasons
- Developers often understand the issues of one phase better during later phases
- It is difficult to assess progress until some things are implemented

Question

How would you improve upon the Waterfall Model?

Incentive Mismatch

- Requirements can create perverse incentives for clients to:
  - Avoid rigorous thinking about cost, change, priorities, and risk
  - Delegate hard design decisions to IT
- It is inexpensive for clients to generate many requirements
- Developers are rewarded for building to requirements, not for refining and removing them
- Schrage argues for **quick prototypes** based on a few (20-25) requirements.[1]

[1] Schrage, Michael. "Never go to a client meeting without a prototype" IEEE Software 21.2 (2004): 42-45.

Prototyping

- Prototype is an incomplete model of eventual system
  - Developed rapidly based on initial requirements
  - Provided to users for evaluation

- User feedback aids refinement and validation of requirements
  - Especially helpful with look-and-feel and user interactions
  - Can also be used to validate an internal design
    - E.g., to assess performance or capacity

See https://retool.com

Prototyping

- Focus should generally be on areas of greatest risk to project

- To produce prototype rapidly:
  - Functionality can be **omitted**
  - Non-functional constraints can be **ignored** (e.g., efficiency, reliability)
  - Existing components can be **reused**
  - "Rapid development" tools and languages can be **exploited**

Prototype Fidelity

- Classical prototypes have **high fidelity** to final product
  - Executable, e.g., interactive mockups of UI

- More generally, prototypes can vary in fidelity
  - Paper prototypes
  - Storyboards

What tradeoffs are connected to different levels of fidelity?

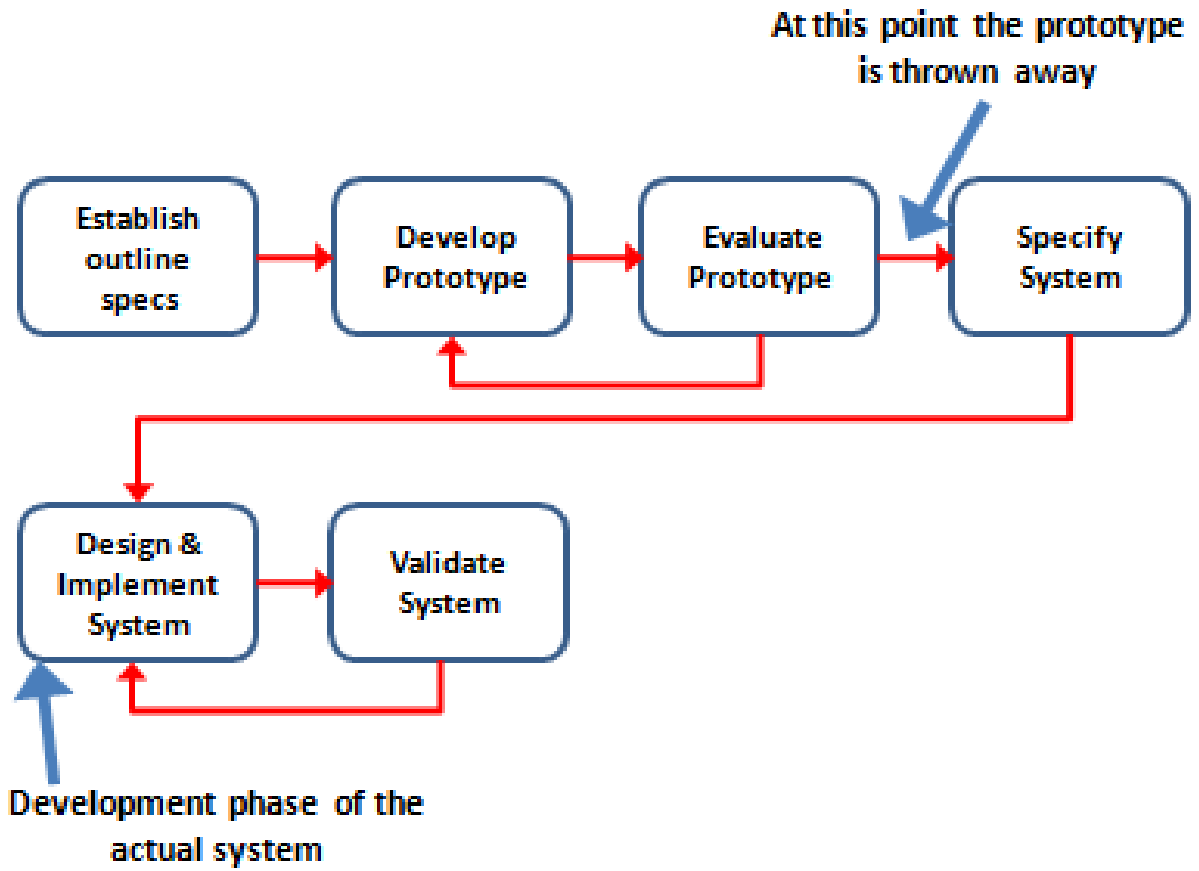
Types of Prototyping

- **Throwaway**: prototype is not built upon

- **Evolutionary**: prototype is iteratively refined and extended to obtain final system
  - Refactoring (restructuring) is necessary to make and keep design coherent

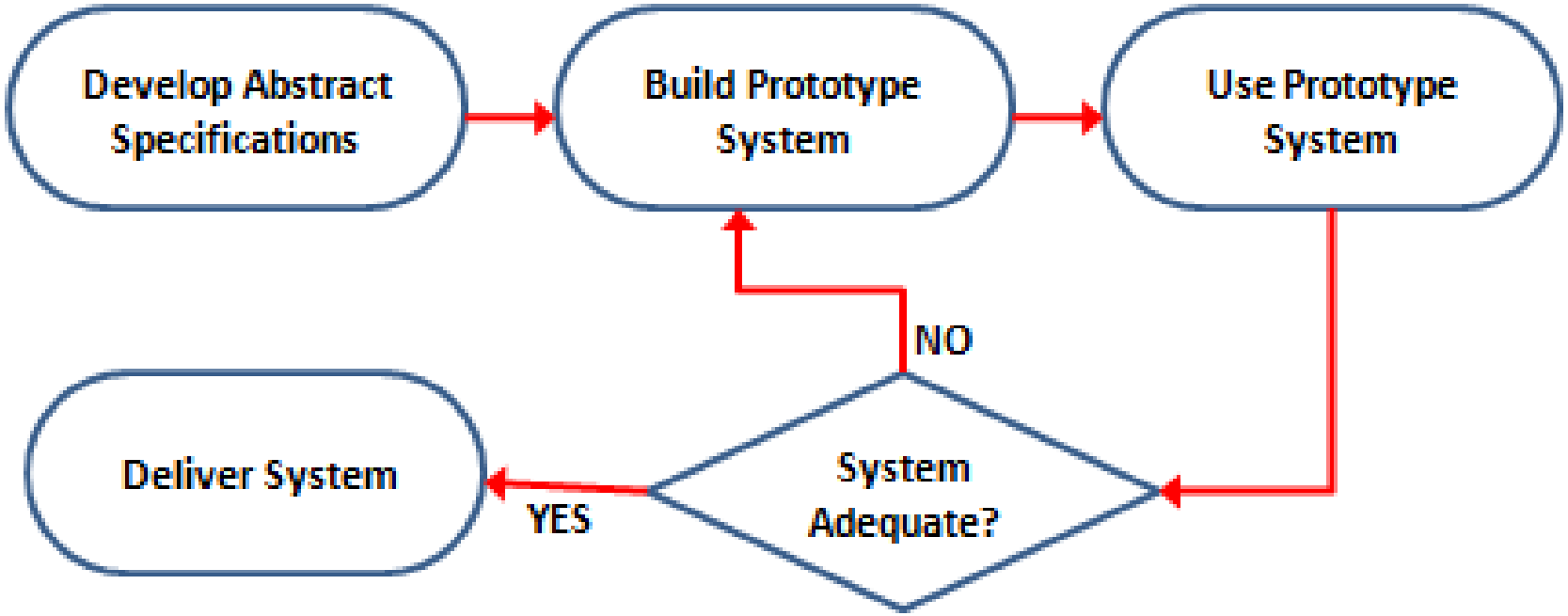
    See www.cs.unc.edu/~stotts/723/refactor/chap1.html

Throwaway Prototyping



https://mahaerss.wordpress.com/about/2-software-engineering-methods/

Evolutionary Prototyping



Operational Prototyping

- This combines throwaway and evolutionary prototyping to achieve rapid results with stability.
- An evolutionary prototype is constructed and made into a baseline using conventional methods
  - Only well-understood requirements are implemented.

Copies of baseline are sent to multiple customer sites along with a trained prototyper.

Davis, Alan M., and Pradip Sitaram. "A concurrent process model of software development." ACM SIGSOFT Software Engineering Notes 19.2 (1994): 38-51.

Risks of Prototyping

- Possible neglect of up-front analysis
- Users may misunderstand purpose of prototype
- Accommodating users may lead to “feature creep”
- Excessive effort may be required
- Possible contractual difficulties

Questions

- Do you see any other potential drawbacks to prototyping?
- How does the internet facilitate prototyping?

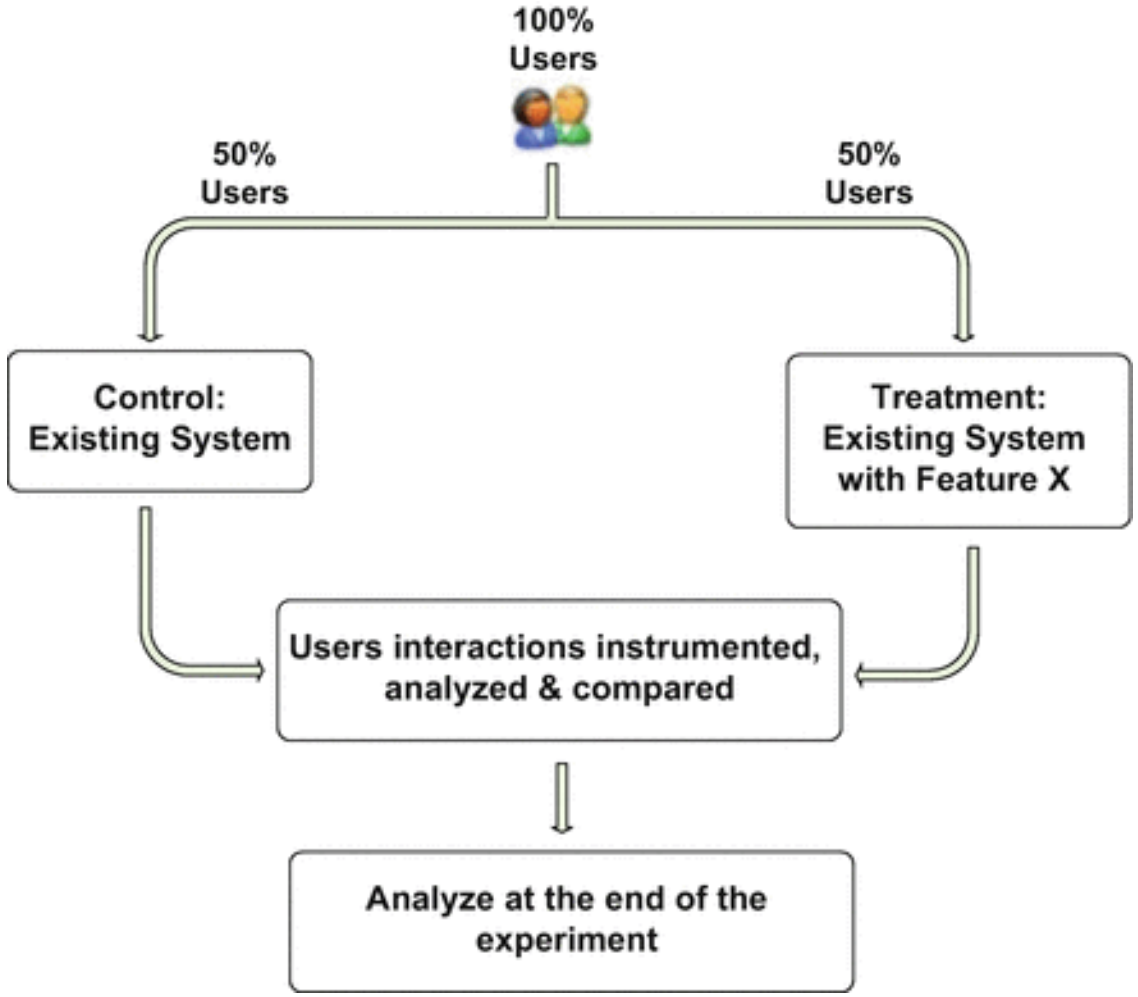
A/B Testing – Online Controlled Experiments

- An experimental comparison of versions of webpage or app to determine which is better
  - Objective overall evaluation criterion (OEC) must be specified
    - e.g., conversion rate, units purchased, revenue, profit, expected lifetime value, or a weighted combination of these
- The versions are assigned at random to different users, persistently

A/B Testing

- Users' interactions with site are **monitored** and key **metrics** computed
- Uses techniques for **design and analysis** of experiments to detect "significant" differences in OEC values between versions

Structure of Online Experiment



Kohavi, Ron, and Roger Longbotham. "Online controlled experiments and A/B tests." Encyclopedia of machine learning and data mining (2015): 1-11.

Example: Evaluating Possible Bing Feature

- Feature allows advertisers to provide multiple links to target site
  - Criterion: increasing average revenue without degrading user engagement

Ads with site link experiment. Treatment (bottom) has site links. The difference might not be obvious at first but it is worth tens of millions of dollars

Why A/B Testing

- "Controlled experiments embody the best scientific design for establishing a causal relationship between changes and their influence on user-observable behavior."

- "When a company builds a system for experimentation, the cost of testing and experimental failure becomes small, thus encouraging innovation through experimentation."

Opportunistic Programming

- Programmers "prototype, ideate, and discover"
- Emphasizes speed and ease of development over maintainability and robustness
  - Often used when individuals construct a program for their own use.
- Involves web foraging and just-in-time learning for
  - Ideas, examples, APIs, code, technical details, etc.

Brandt, Joel, et al. "Two studies of opportunistic programming: interleaving web foraging, learning, and writing code." *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2009.

Characteristics of Opportunistic Programming

- Build from scratch using high-level tools
- Add new functionality via copy-and-paste
- Iterate rapidly
- Consider code impermanent
- Unique debugging challenges

Question

Do you see any potential problems with opportunistic programming?

Opportunistic Programming Issues

- Rights to intellectual property
- Plagiarism
  - Not permitted for CSDS 393/493 projects!
- Reliability, security, maintainability are hard to ensure

Incremental Delivery

- Developed and delivered in series of increments
- Each increment provides a subset of the system functionality
- Services are allocated to increments based on customer's priorities and risks
- A conventional process is applied to each increment

Advantages of Incremental Delivery

- Client can exploit product functionality sooner
- Client can adapt to product gradually
- Developer gets earlier feedback than with waterfall model
- Requires planning for future enhancements
- Highest priority services get most testing

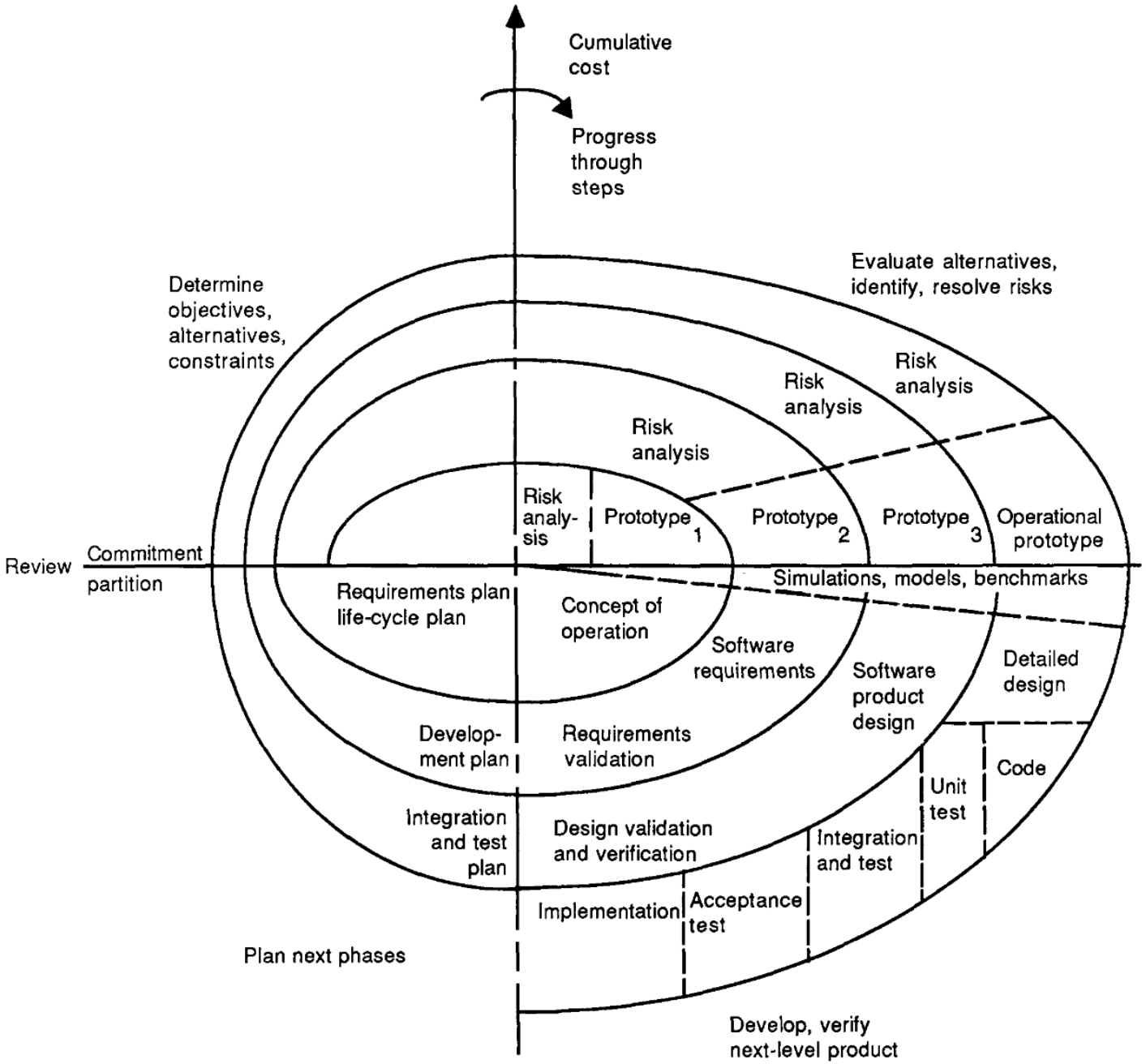
Risks of Incremental Delivery

- It may be difficult to integrate later builds with early ones
  - Why?
- It can degrade into *build-and-fix*

Spiral Model [Boehm 1986]

- Assumes that risk management is a paramount issue in software development
- Development process is represented by a spiral
- Each cycle represents a phase with four parts:
  1. Setting objectives
  2. Risk analysis and mitigation
  3. Development and validation
  4. Planning for next phase

Spiral Model



Boehm, Barry W. "A spiral model of software development and enhancement." *Computer* 21.5 (1988)

Analysis of Spiral Model

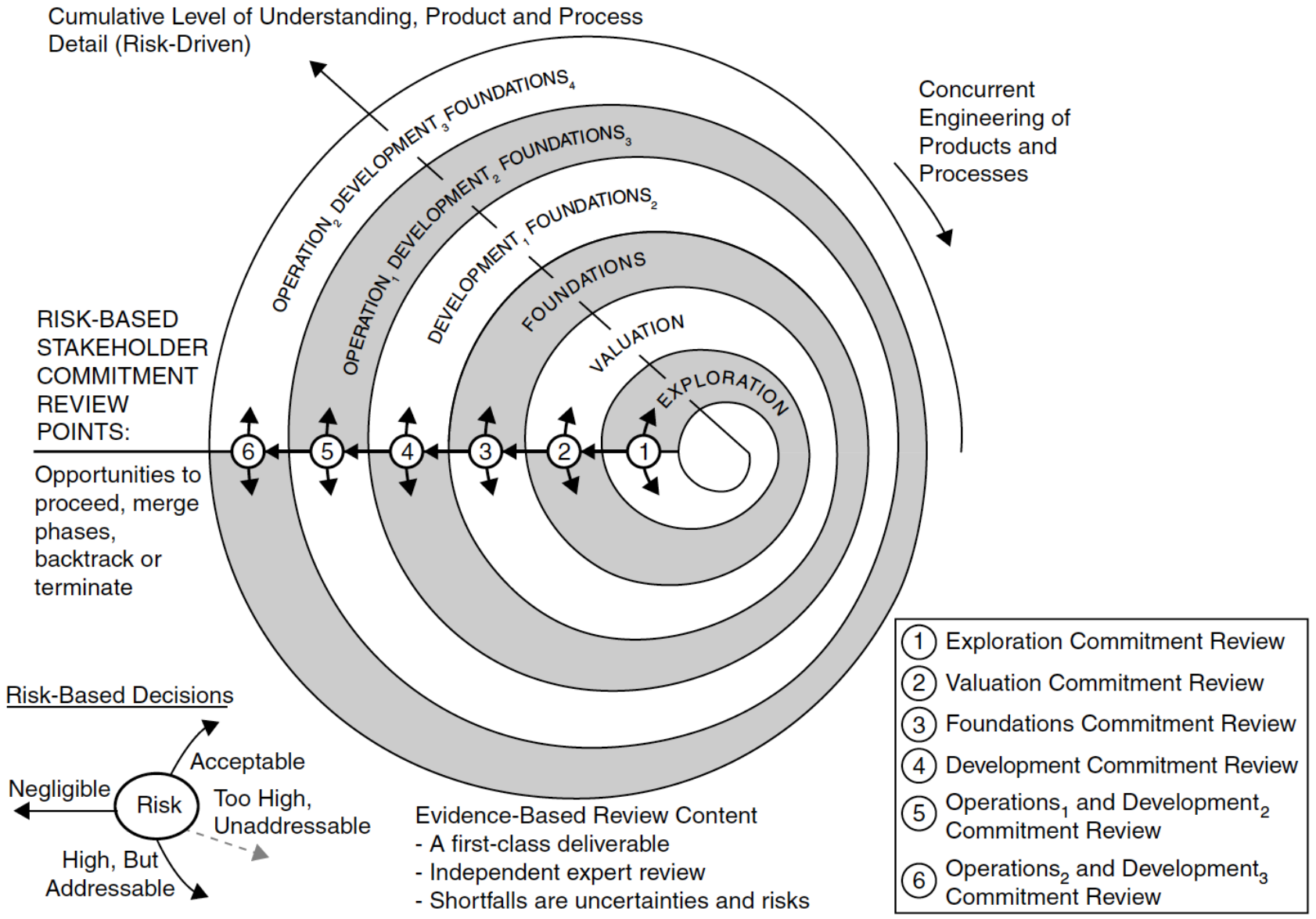
- Best suited to projects with:
  - Large scale
  - High risk
  - Ample resources and time
  - Client and developers within same organization

Evolution of the Spiral Model

- Boehm later described it as a risk-based "process model generator"
  - Other models are special cases that fit the risk patterns of certain projects

- **Recent revision**: Incremental Commitment Spiral Model

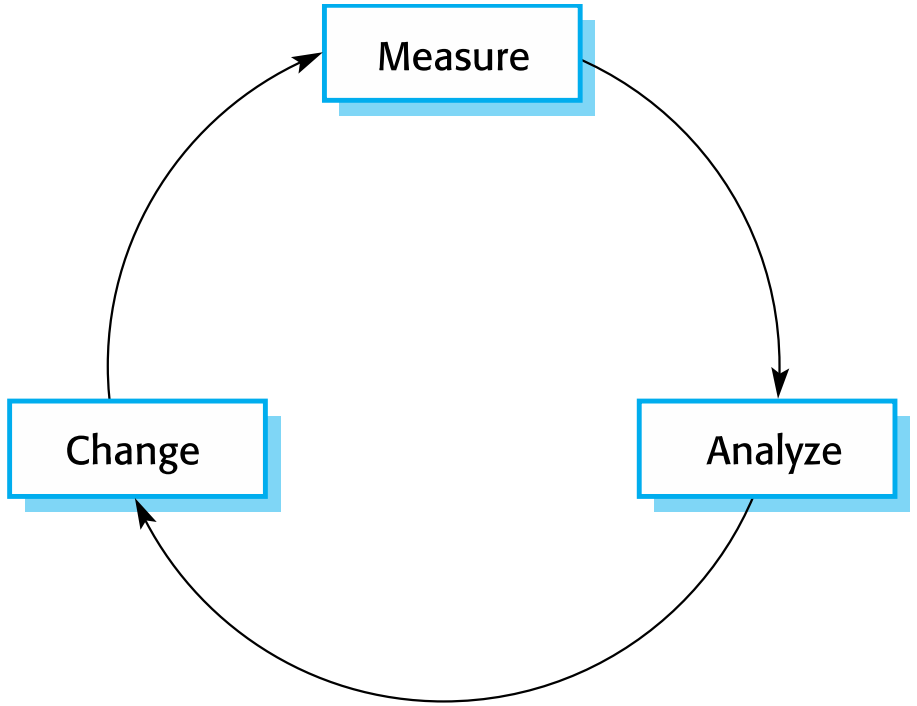
Incremental Commitment of Spiral Model



https://boehmcsse.org/tools/icsm/

Process Improvement

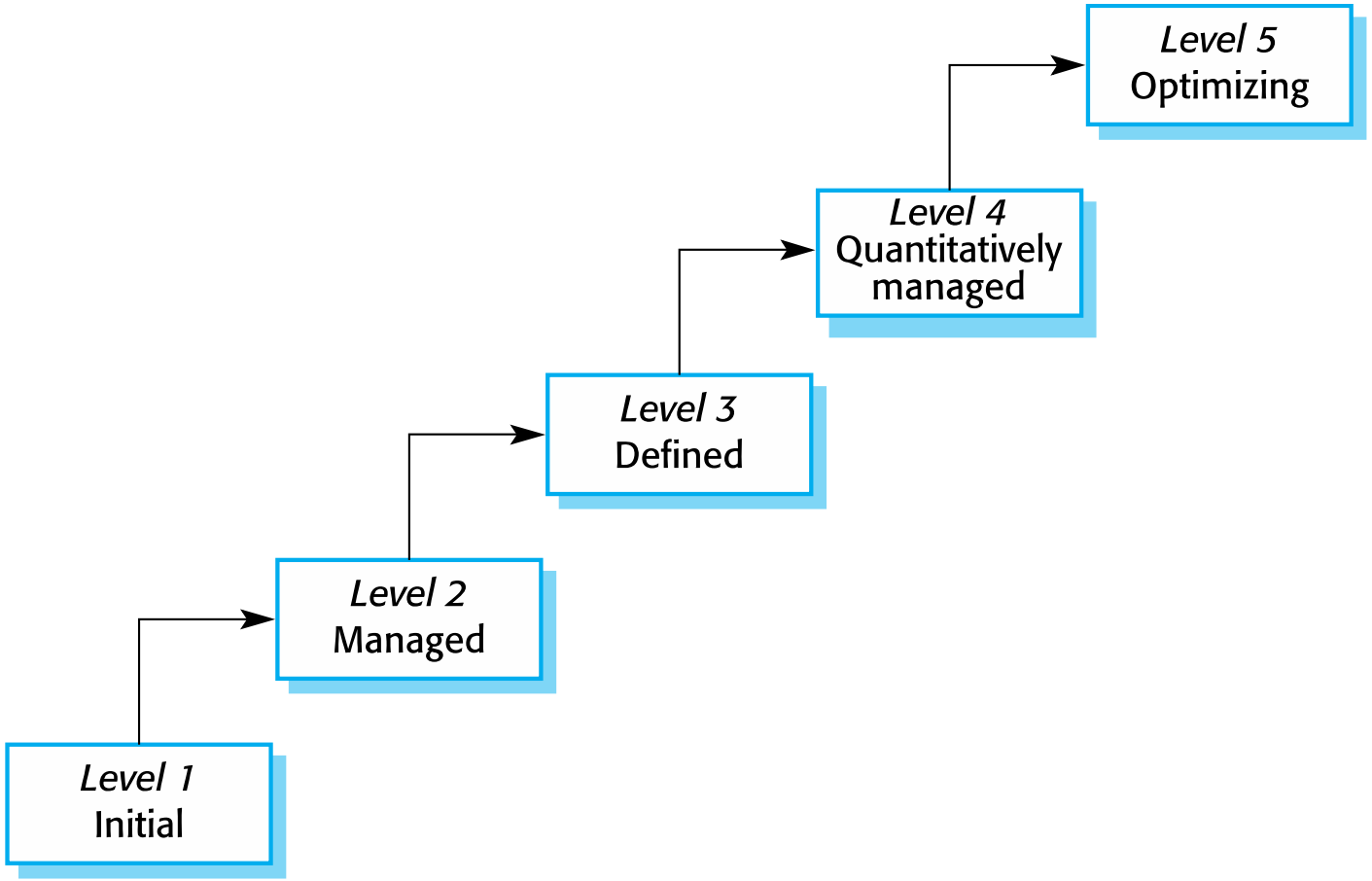
- Understanding existing processes
- Changing these processes for
  - Increase product quality
  - Reduce costs and development time



Process Metrics

- Time taken for process activities to be completed
  - E.g. Calendar time or effort to complete an activity or process.
- Resources required for processes or activities
  - E.g. Total effort in person-days.
- Number of occurrences of a particular event
  - E.g., Number of defects discovered.

Capability Maturity Levels



Further Readings

- GitHub flow
- Schrage, Michael. "Never go to a client meeting without a prototype" IEEE Software 21.2 (2004)
- Kohavi, Ron, and Roger Longbotham. "Online controlled experiments and A/B tests." Encyclopedia of machine learning and data mining (2015)