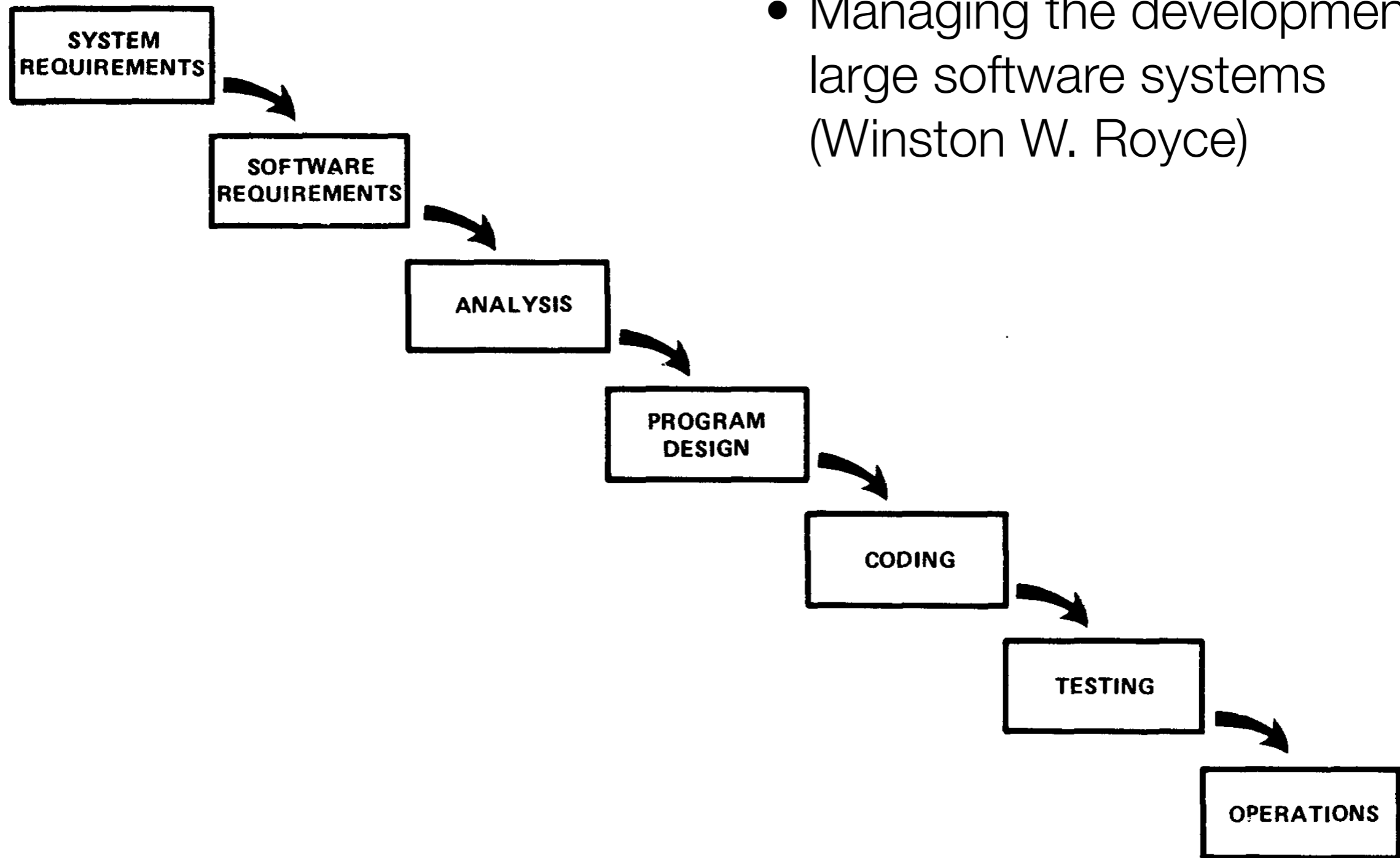# Software Development Processes

The Processes

- **Software Development Process:** a structured and progressive refinement from idea to an actual system

- Processes
  - Waterfall
  - Prototype
  - Incremental
  - Spiral
  - V-Model
  - Cleanroom
  - RUP
  - XP
  - SCRUM
  - RAD
  - ASAP
  - MSF
  - DFDM

- Waterfall defined by Royce (seventies)

- Introduced to address the "software crisis"

- New processes proposed to:
  - Increase flexibility in the organization of development activities
  - Improve:
    * User satisfaction (building systems that are closer to user needs)
    * Efficiency (building systems faster)
    * Time and costs (being more reliable with estimations)
    * Quality

- Heavy-weight vs. Agile
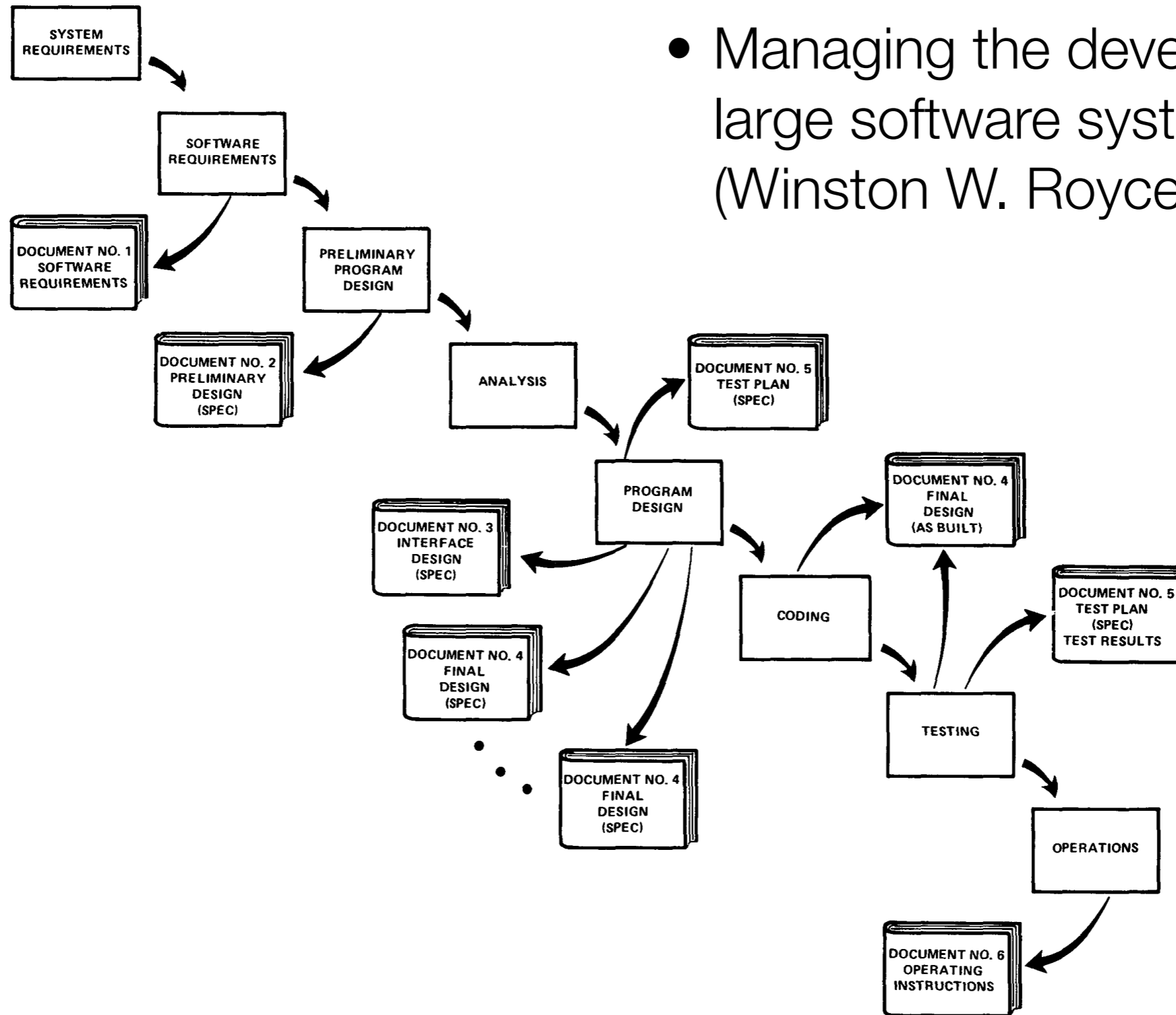
Waterfall Model

SYSTEM REQUIREMENTS

SOFTWARE REQUIREMENTS

ANALYSIS

PROGRAM DESIGN

CODING

TESTING

OPERATIONS

- Managing the development of large software systems (Winston W. Royce)

Waterfall Model

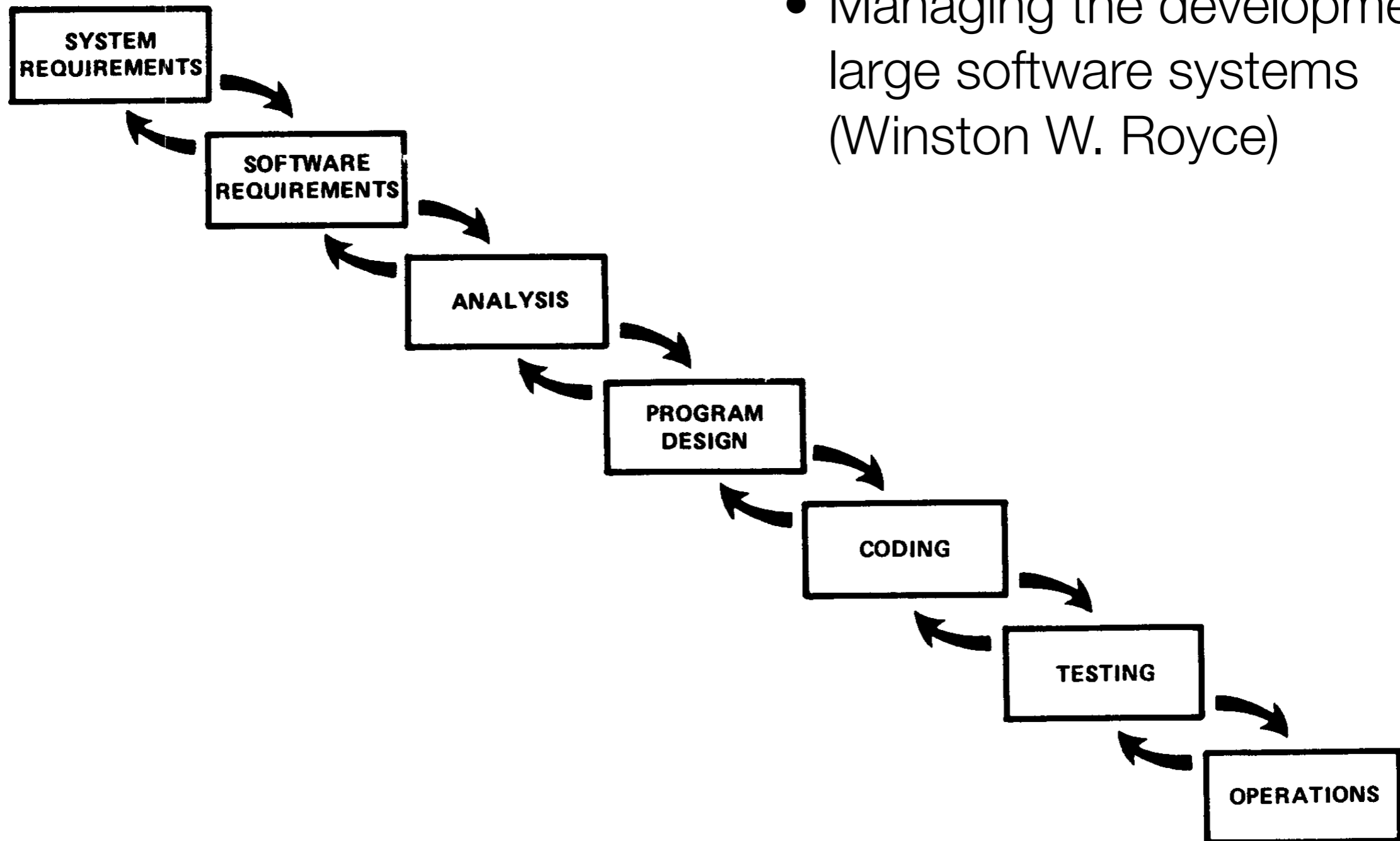- Managing the development of large software systems (Winston W. Royce)

Waterfall Model

- Inflexible partitioning of the project into distinct stages

- This makes it difficult to respond to changing customer requirements

- This model is only appropriate when:
  - The requirements are well-understood and/or
  - The level of formality is high (e.g. it is essential to "freeze" the requirement document)

Waterfall Model

- Managing the development of large software systems (Winston W. Royce)

The diagram shows boxes connected by arrows in a cascading pattern: SYSTEM REQUIREMENTS → SOFTWARE REQUIREMENTS → ANALYSIS → PROGRAM DESIGN → CODING → TESTING → OPERATIONS

Waterfall Variations

- **Sashimi waterfall:** activities are allowed to overlap

- **Waterfall with sub projects:** implementation of different components proceeds in parallel

- **Waterfall with risk reduction:** an initial risk analysis helps mitigate risks in later phases of implementation

V-Cycle

- Structural support for backtracking

- Focus on testing

- German standard

Rational Unified Process
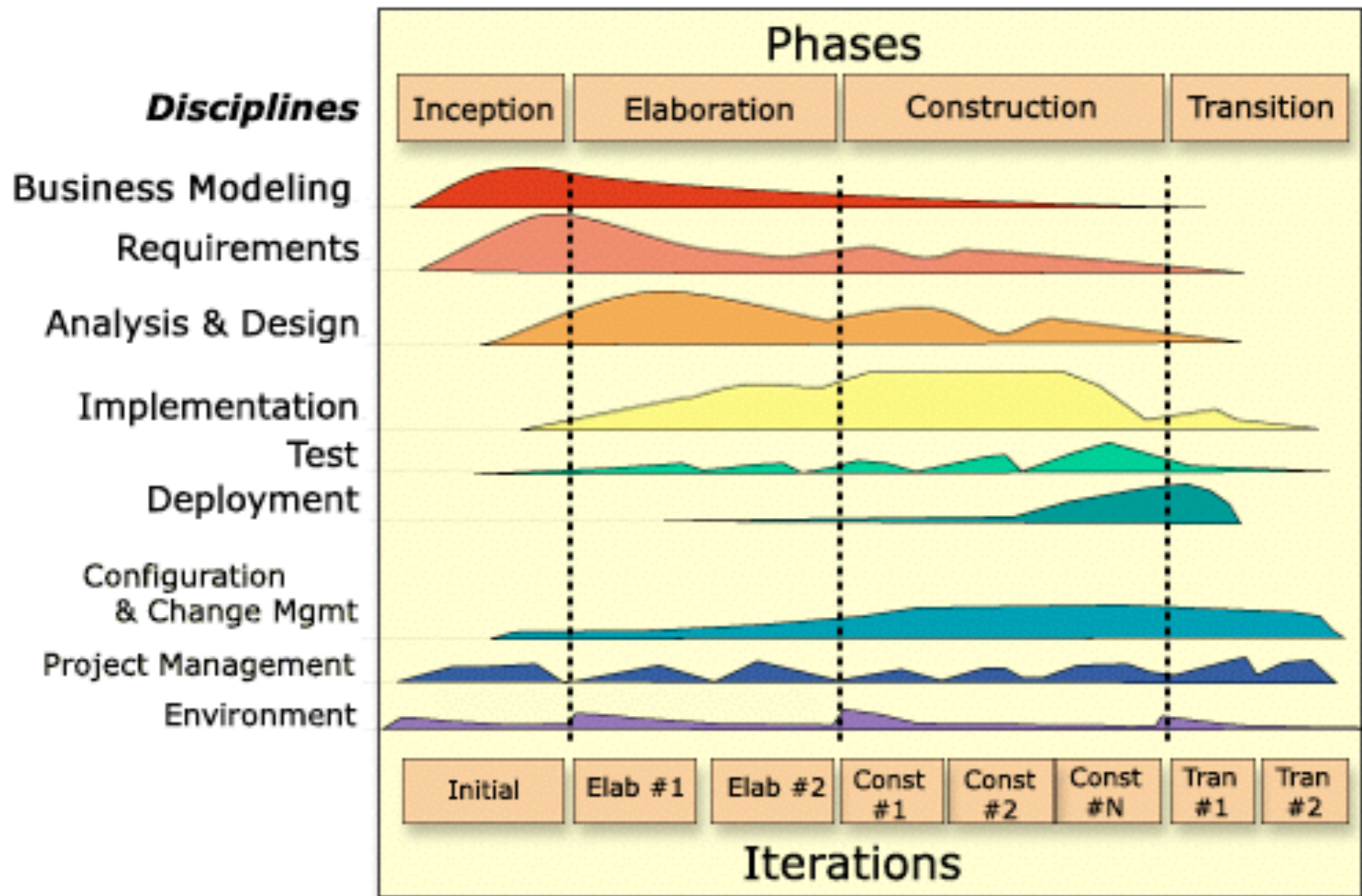
Rational Unified Process

- Process introduced by Rational in the eighties
  (the same company of UML) [Rational is now IBM]

- Process organized in two dimension:
  - phases, organized in iterations, correspond to different levels of maturity
  - workflows, focusing on a specific software development concern

- Phases are organized in iterations

- Workflows are overlapping and characterized by levels of intensity

Rational Unified Process Best Practices

- Six main practices define guiding principles of RUP:
  - Develop software iteratively
  - Manage requirements (including evaluation of product alternatives)
  - Use component-based architectures (robust components)
  - Visually model software (simple and unambiguous representation to build a shared vision)
  - Verify software quality
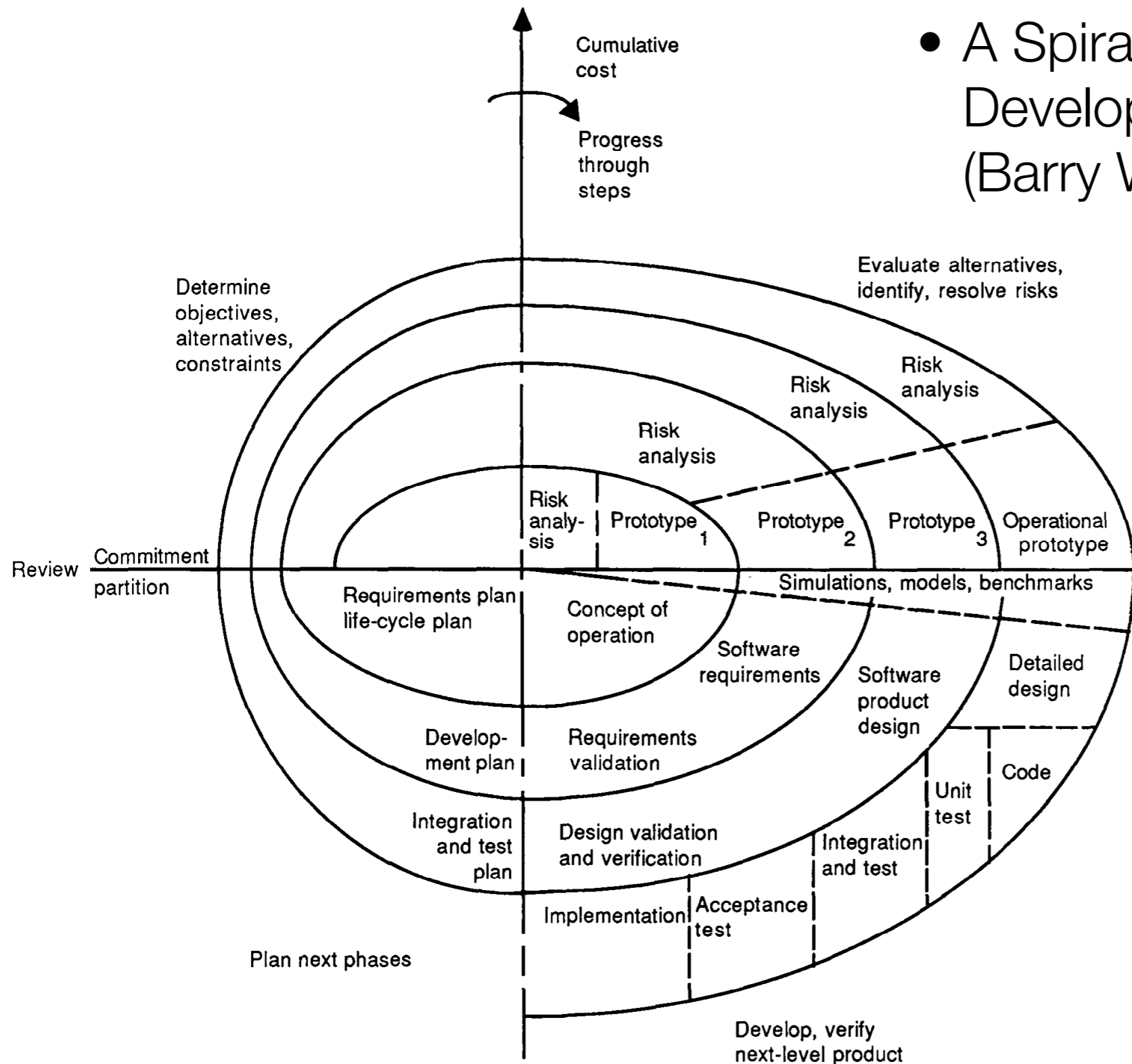 - Control changes to the software (both for quality and management)

Open Unified Process
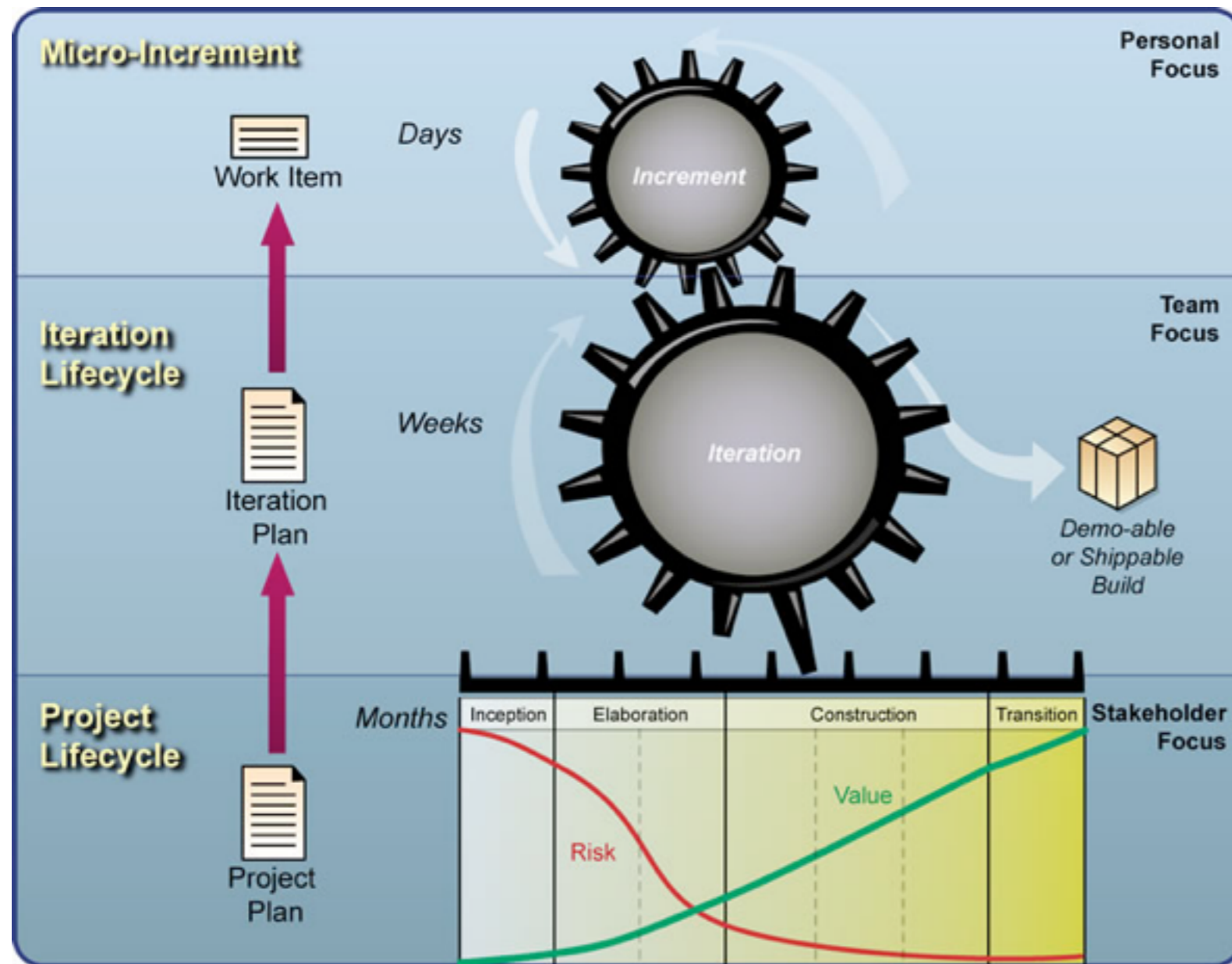
- The evolution of RUP

- Open source

- "Agile"

- http://epf.eclipse.org/wikis/openup/

Spiral

- A Spiral Model of Software Development and Enhancement (Barry W. Boehm)

Spiral

- Defined by Barry W. Boehm (end of the '80s)

- Iterative: software is developed in cycles

- Each loop in the spiral represents a phase in the process.

- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required

- Alternative and risk-aware: first phases include an evaluation of possible alternatives and an assessment of the risks

- The paper includes a list of common project risks (*)(not only process, also practices)

(*) which we will look at during the Risk Management lessons

Spiral

- Advantages
  - Alternative and Risk driven
  - It accommodates different software development practices (among which reuse and automatic code generation)
 - Intrinsically fit for software evolution (maintenance is another loop in the spiral)

  - Matching to contract software
  - Difficulties in coming out with estimations at the beginning
  - Flexibility

- Disadvantages

Prototype Approach

- Once the high level requirements are fixed, a prototype of the application is developed (e.g. the GUI) and evaluated with the client

- Breadth and depth of the prototype
  - **Horizontal prototype:** focus on the application
  - **Vertical prototype:** focus on a specific function

- Types of prototypes:
  - **Throw-away:** the prototype demonstrates the system (but it is not the system!)
  - **Evolutionary:** the prototype evolves to become the system

Prototype Approach
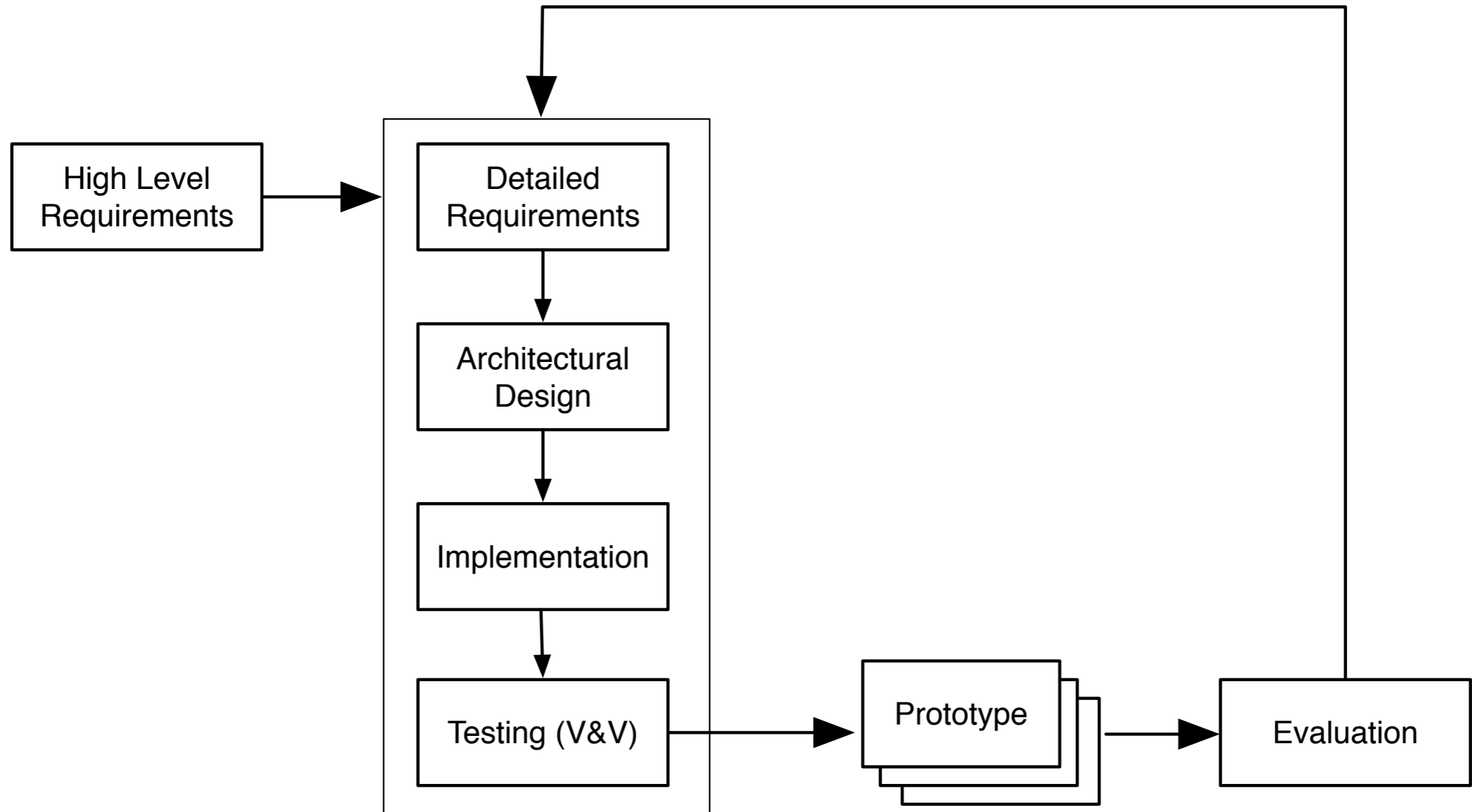
Prototype Approach

- **Advantages**
  - Shared view on requirements: the client has a better understanding of the final product and can provide more specific comments.
  - Difficult and risky components can be developed early to verify the feasibility of critical parts of the system

- **Disadvantages**
  - Lack of process visibility
  - Re-work (and frustration)
  - The evolutionary approach leads to poor architectures (because they are built with incomplete information)

- Applicability
  - For small or medium-size interactive systems
  - For parts of large systems (e.g. the user interface)
  - For short-lifetime systems

Incremental

- Development and delivery is broken down into increments

- Each increment delivers part of the required functionality

- Requirements are prioritized and the highest priority requirements are included in early increments

- Once the development of an increment is started, the requirements are frozen

- Requirements for later increments can continue to evolve

# Incremental
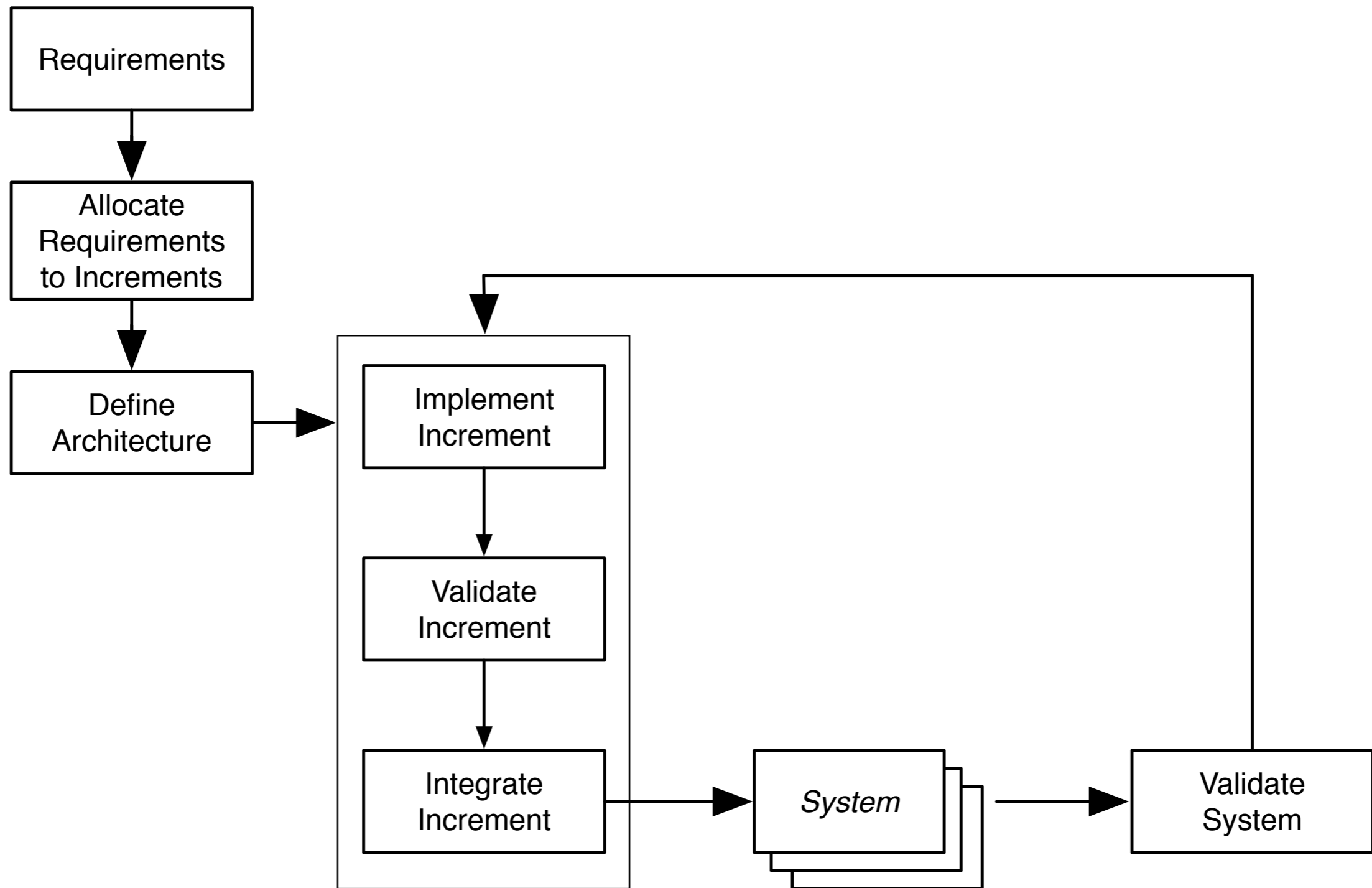
Incremental

- Advantages
  - System functionality is available earlier and customer does not have to wait as long
  - Early increments act as a prototype to help elicit requirements for later increments
  - Lower risk of overall project failure
  - The highest priority system services tend to receive the most testing

- Disadvantages
  - System architecture tends to degrade (as with other evolutive approaches)
  - Some increments might cause significant re-work

Clean Room Software Engineering

- A process for zero-defect software

- Based on formal methods and statistical testing

- Quick and clean … an incremental done right
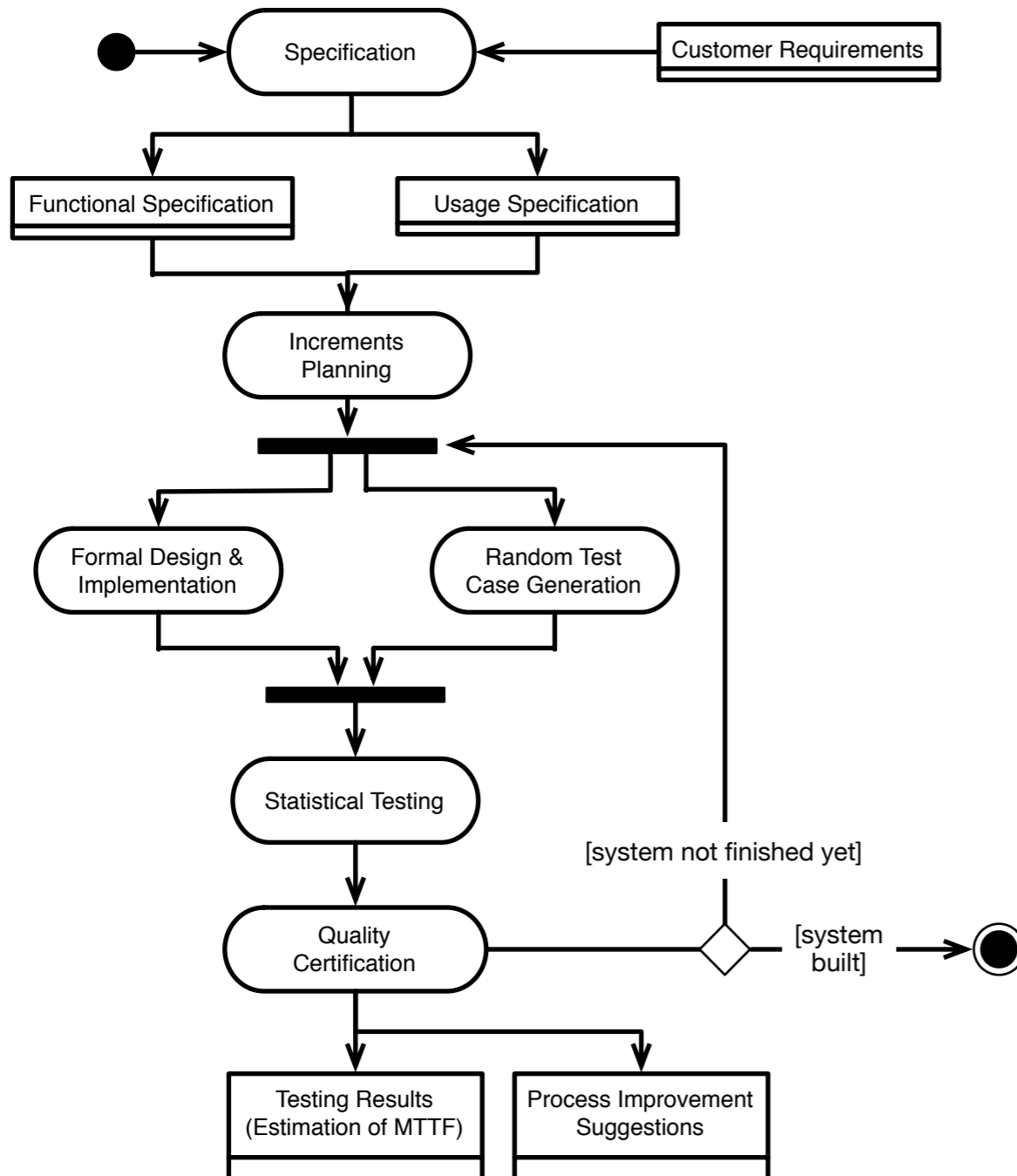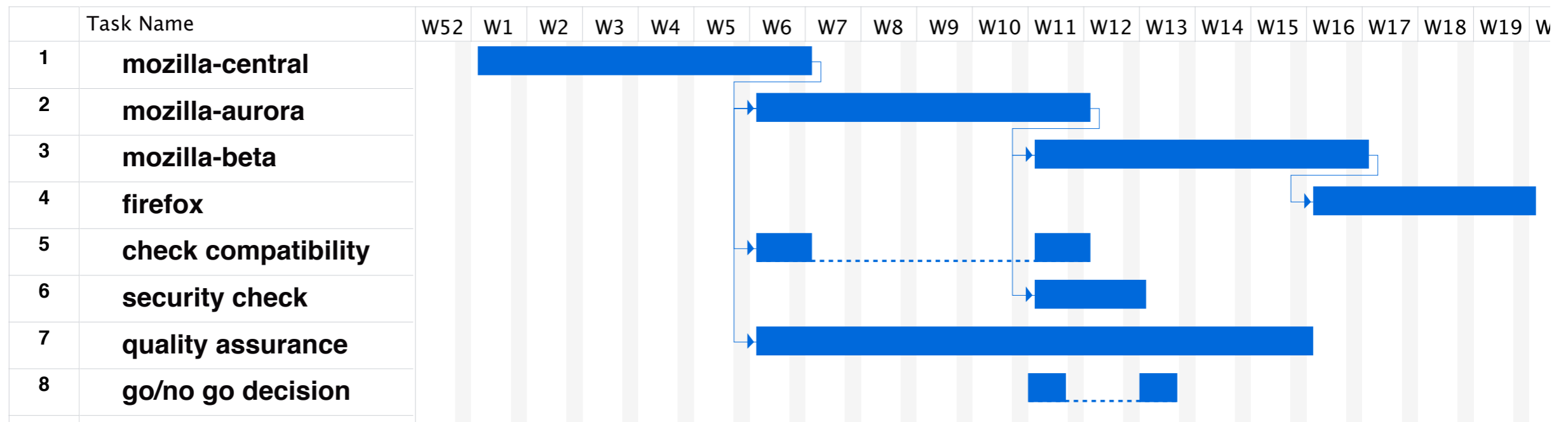
Open Source Development Process

- The development of open source software has various distinguishing features:
  - The project often starts from an existing code based (sometimes large)
  - Community of developers working on a volunteer-basis (not all); with different experience; geographically distributed
  - Some challenges: involvement with different roles; quality control; difficult to plan releases

Open Source Development Process
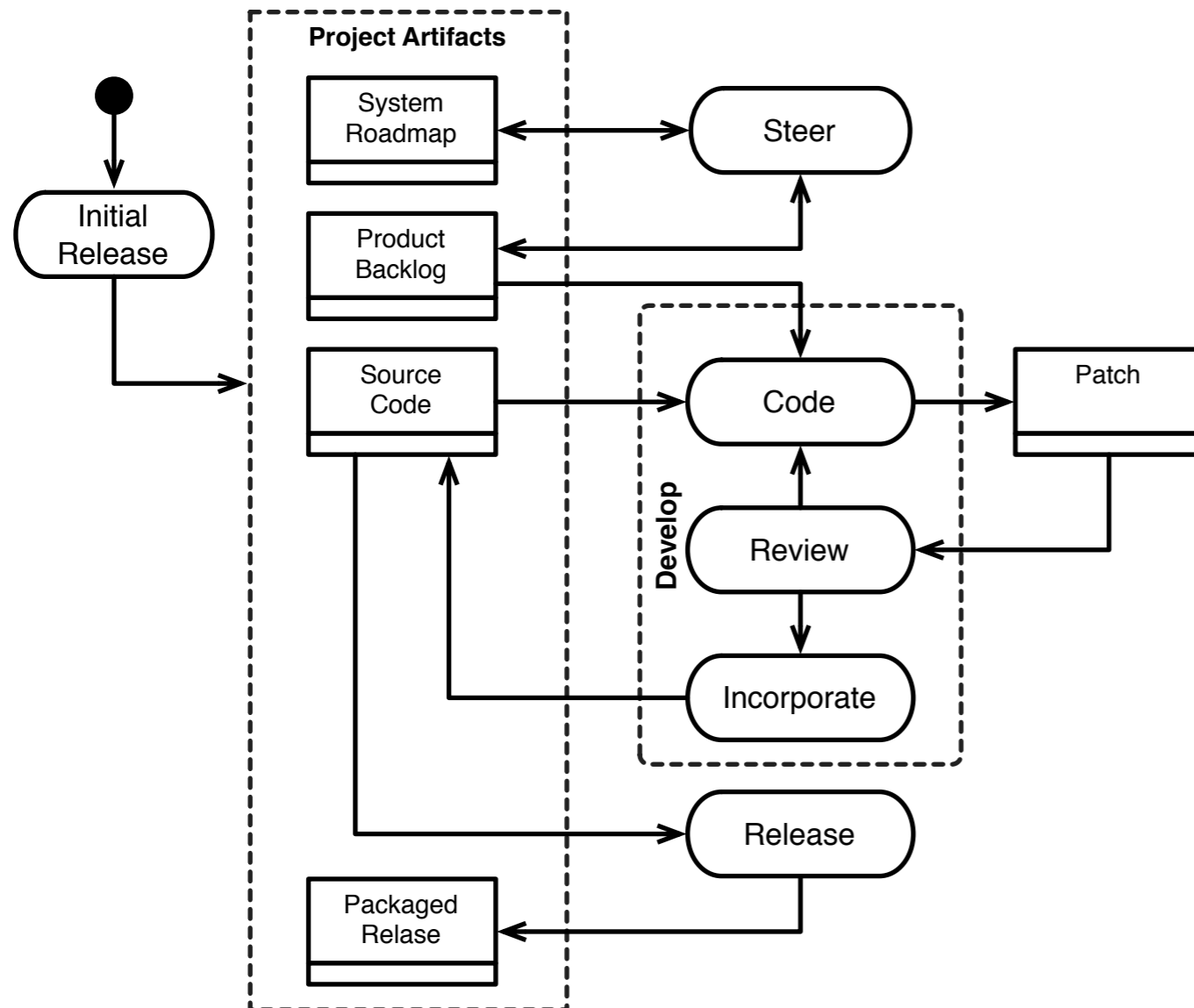
- General model:
  - A roadmap defines software evolution
  - Coding starts from a product backlog and is released as a patch
  - Code review ensures quality

- Two models:
  - fix features (deliver when ready)
 - deliver at fixed dates (with the features you have)

Firefox Release Schedule

	Task Name	W52	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14	W15	W16	W17	W18	W19	W
1	**mozilla-central**																					
2	**mozilla-aurora**																					
3	**mozilla-beta**																					
4	**firefox**																					
5	**check compatibility**																					
6	**security check**																					
7	**quality assurance**																					
8	**go/no go decision**																					

Some Additional Concerns

Project Management/SW Development

- All development processes need to integrate development and management workflows

- For instance:
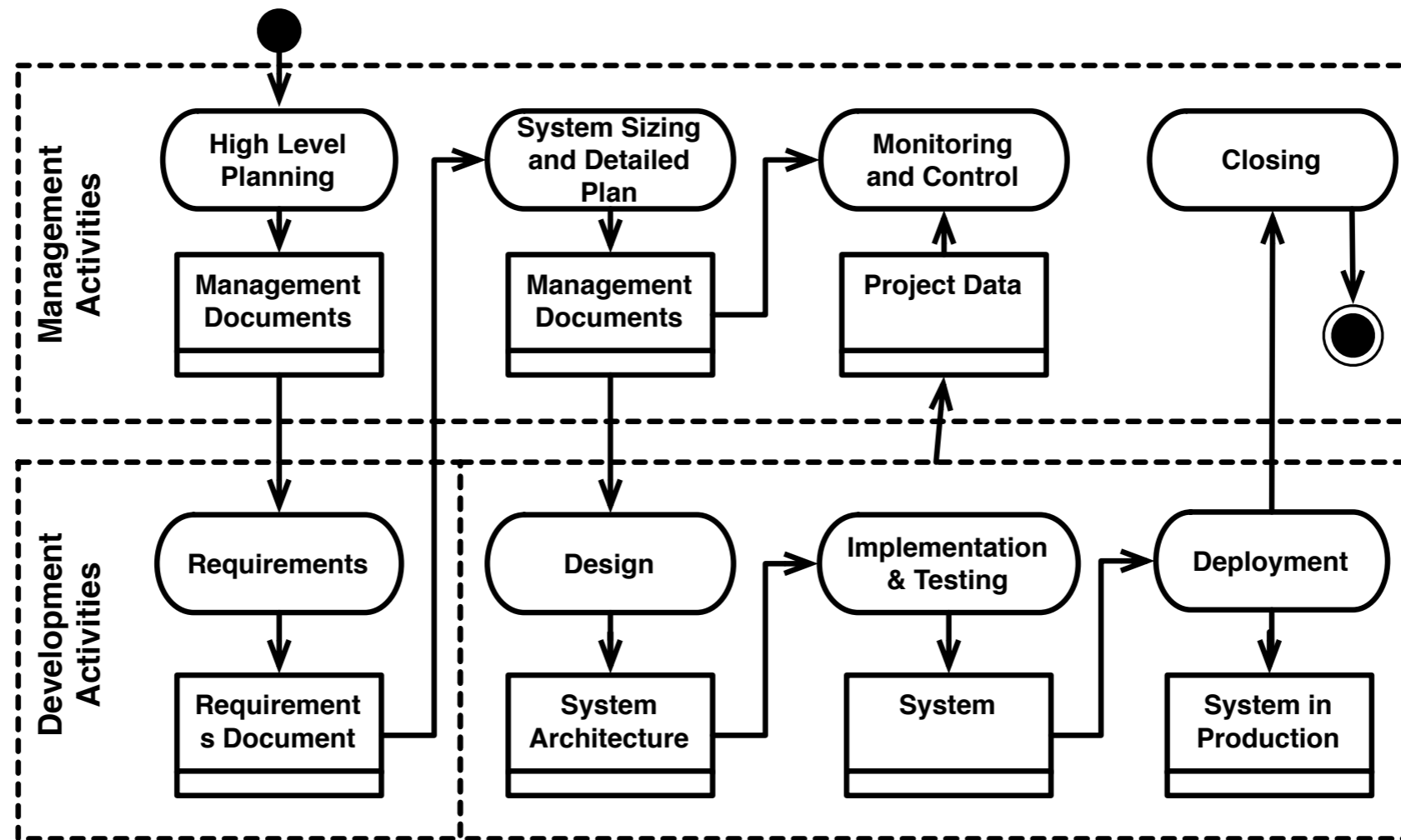
System Organization and Decomposition

- Complex systems are structured in more elementary components

- Different strategies:
  - **Top-down decomposition** (system/component/item): the development of upper levels is achieved by composing more elementary components
  - **Bottom-up Composition** (different components can be developed independently and then integrated)
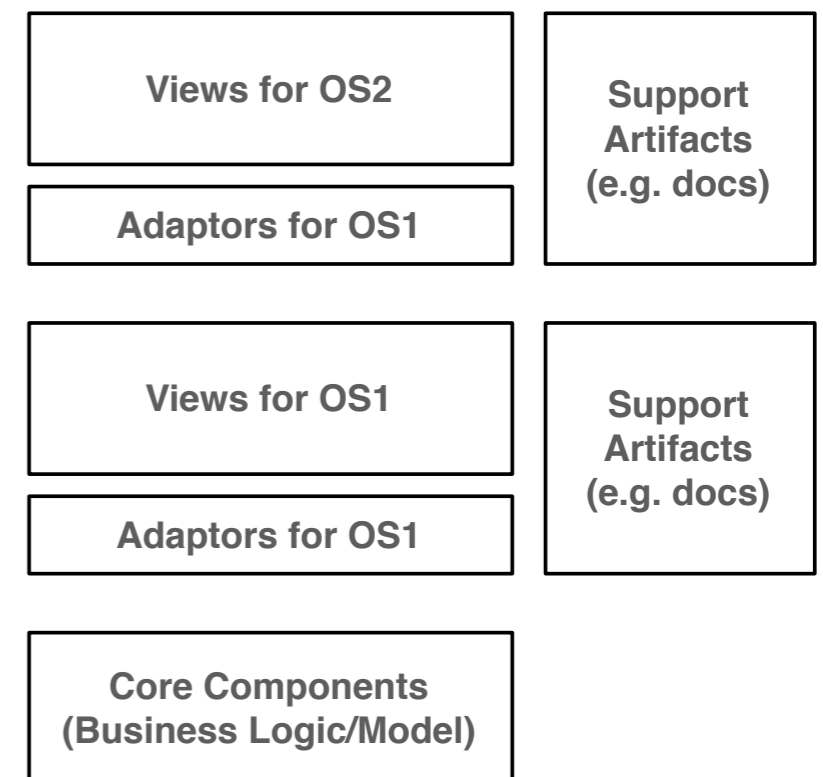
Hierarchical decomposition
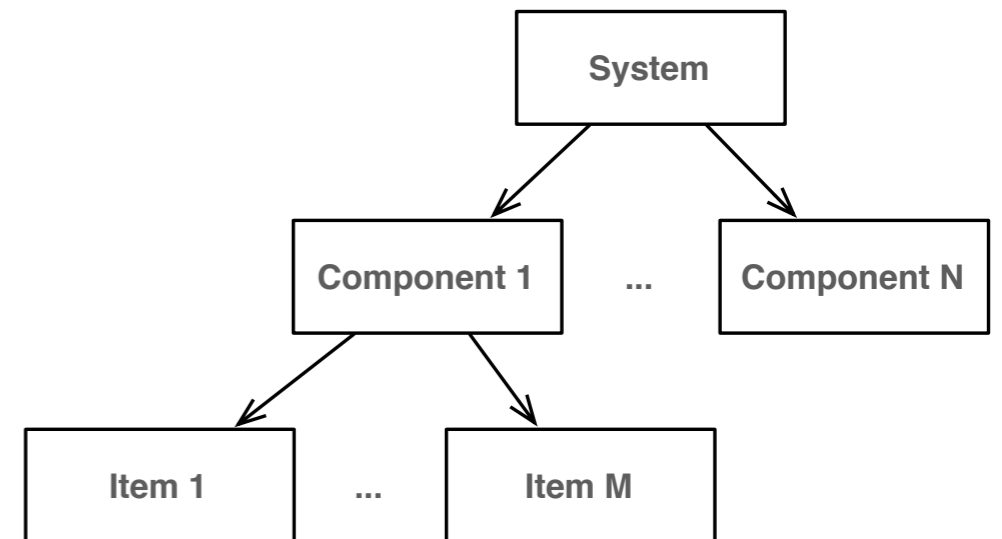
*Components*
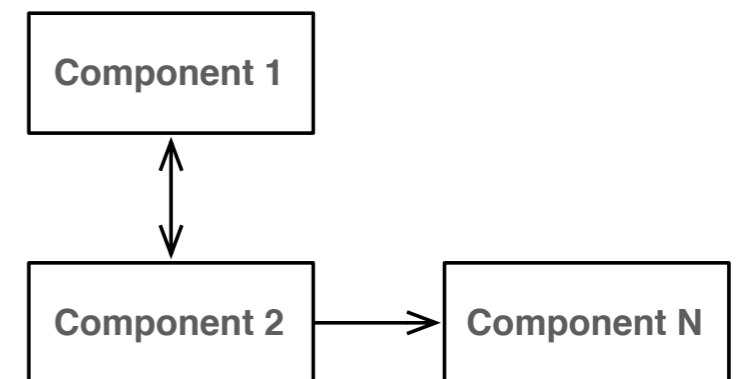
Product Families

- Issues
  - Operating System
  - User experience on different platforms
  - Devices (screen sizes, capabilities)
  - Available technologies

- Approaches
  - Single architecture (e.g. Java)
  - "Adaptable" components (e.g. responsive designs)
  - Different products with shared components
  - Independent development with common interfaces/APIs

Different products with shared components

Views for OS2	**Support Artifacts (e.g. docs)**
Adaptors for OS1	

Views for OS1	**Support Artifacts (e.g. docs)**
Adaptors for OS1	

Core Components (Business Logic/Model)

Some Consequences

- For complex systems:
  - software development activities are "recursive" and carried at increasing levels of granularity
  - software development is composed of various development processes running in parallel
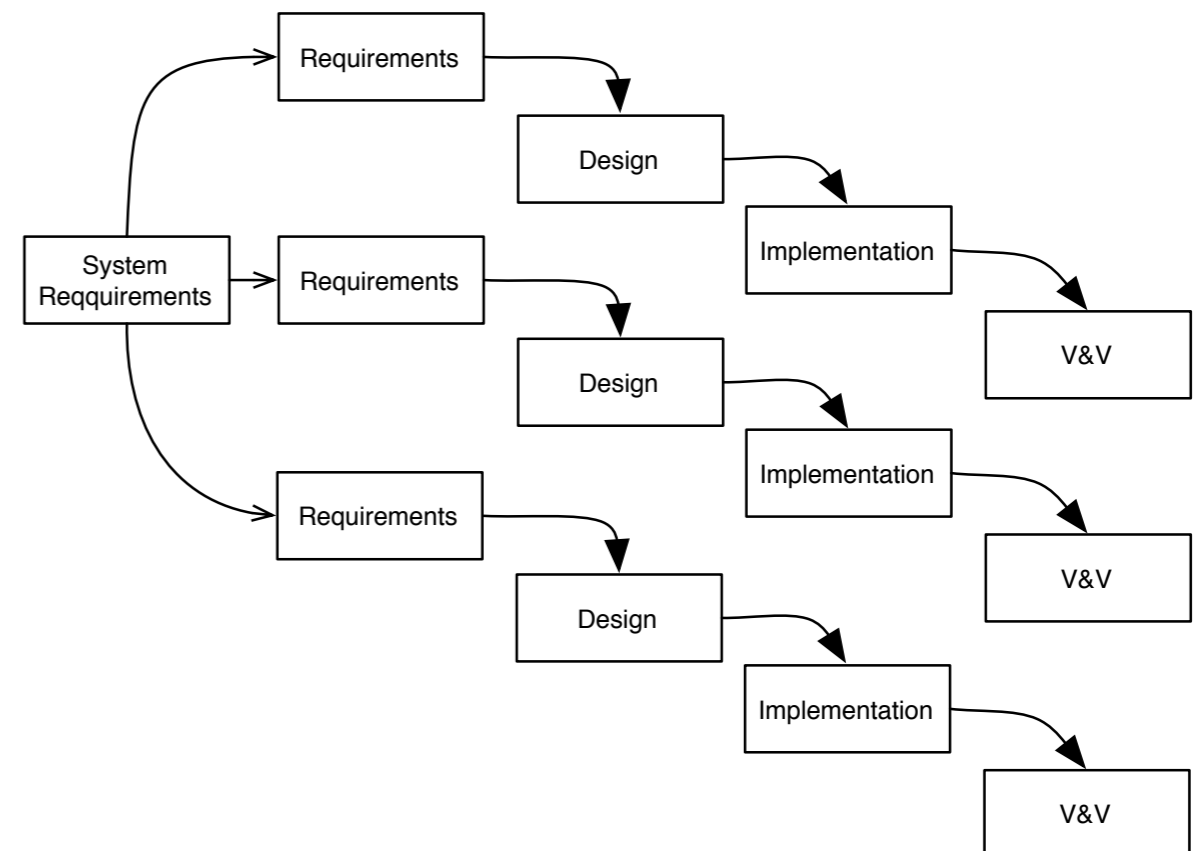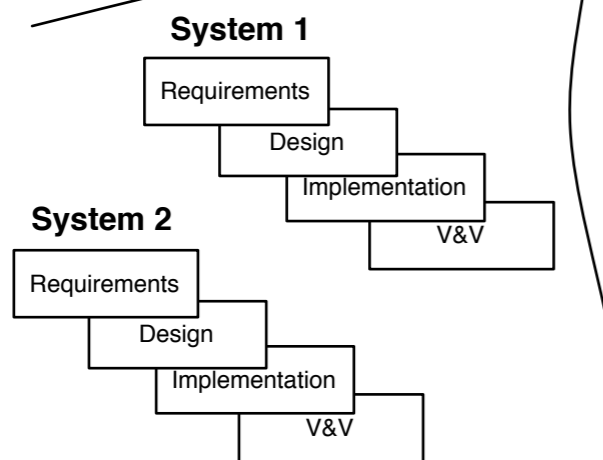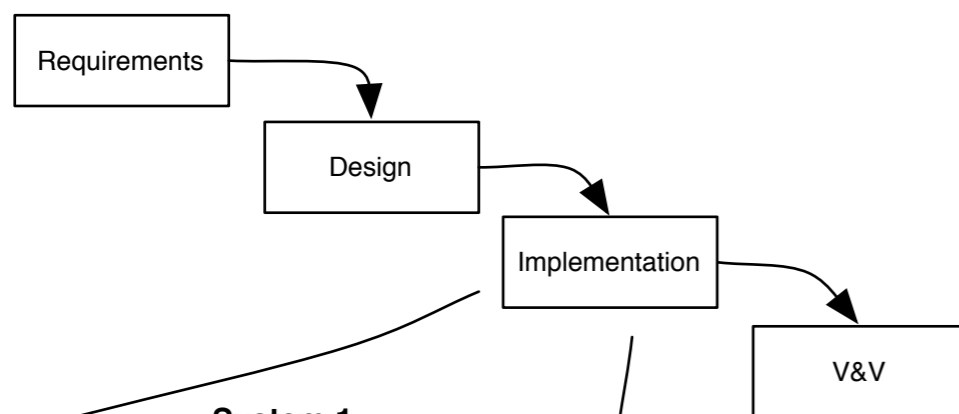
Exercise

Customizing a Process

- We want to build a web application for managing the books of small businesses

- Questions
  - What process?
  - What activities?
  - What products?