Software Process Models

Software Engineering Process Models

- *Process Model*: Simplified, abstract description of how a software project conducts its activities
  - Specification (Requirements Capture)
  - Software Development (Design and Programming/Implementation)
  - Verification and Validation (Quality)
  - Evolution (Maintenance)

Software Engineering Process Models

- *Process Model*: Simplified, abstract description of how a software project conducts its activities
  - Specification (Requirements Capture)
  - Software Development (Design and Programming/Implementation)
  - Verification and Validation (Quality)
  - Evolution (Maintenance)

- We will mention two models and focus on the second
  - Waterfall
  - Incremental Development (agile)

Caution regarding Process Models

- Both Waterfall and Incremental Development have evolved many adaptations. In CS471, we'll use:
  - Waterfall process as defined in *Software Engineering 10th Edition*
  - Incremental Development as defined in *The Elements of Scrum*

- Your mileage may vary!

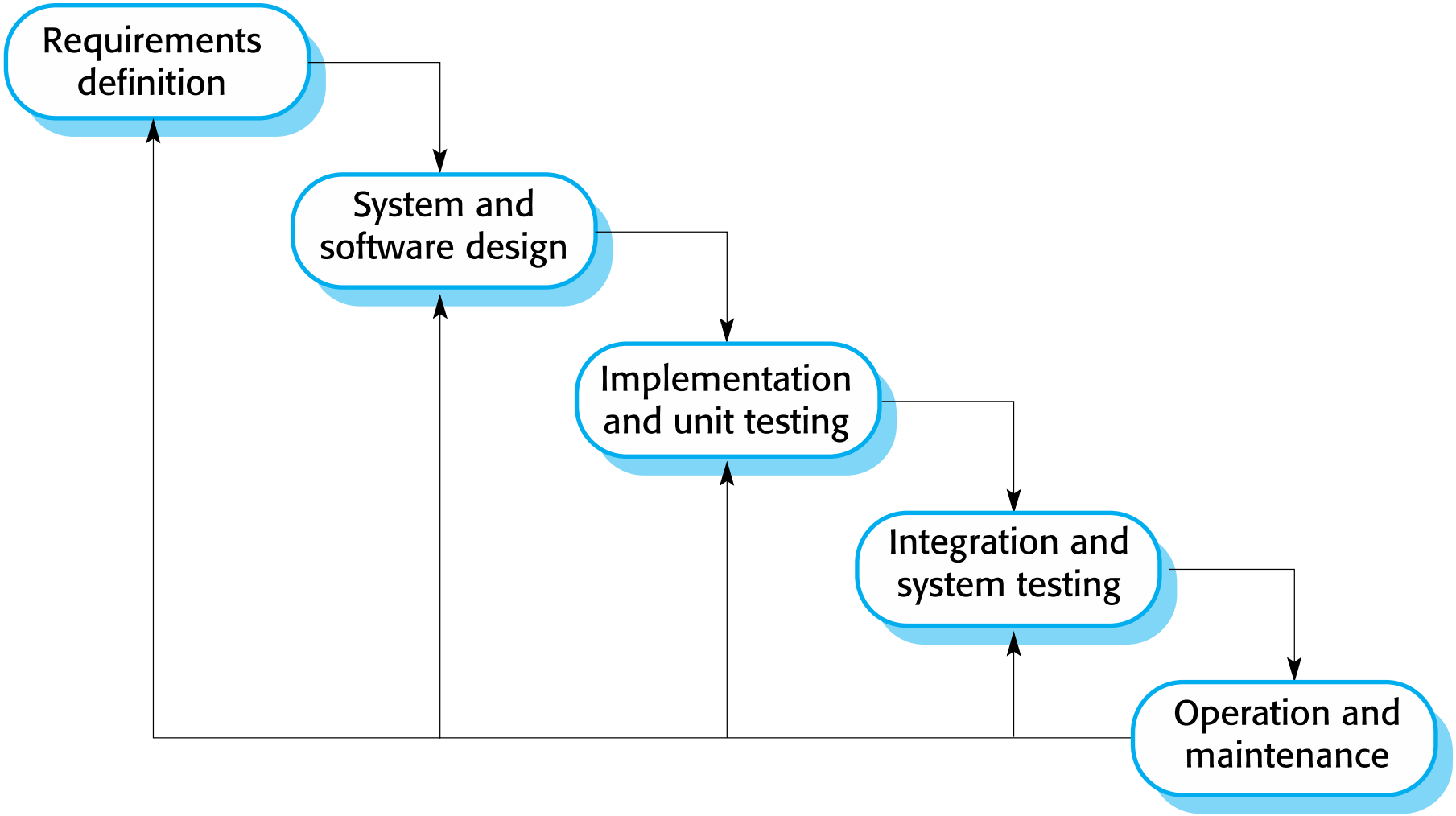
Waterfall vs. Agile

- Waterfall Model (1970)
  - Original process model
  - Plan and make decisions as soon-as-possible
  - Results in long-range plans

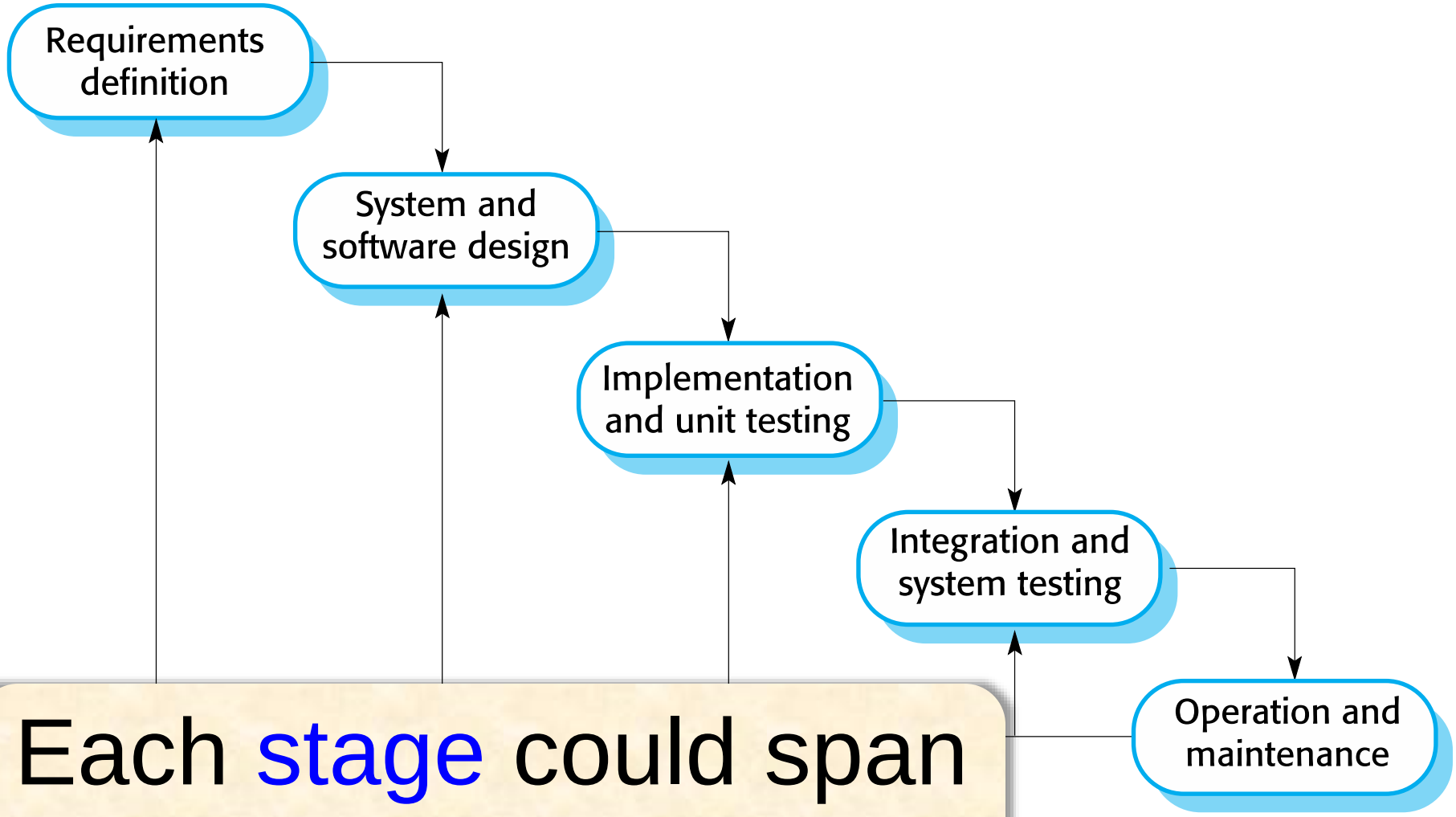
- Agile (Incremental Development) Model (’90s)
  - Plan and make decisions as late-as-possible
  - Results in short-term planning horizons
  - Most popular current model

The Waterfall Process Model

A Waterfall Process Model [Sommerville]

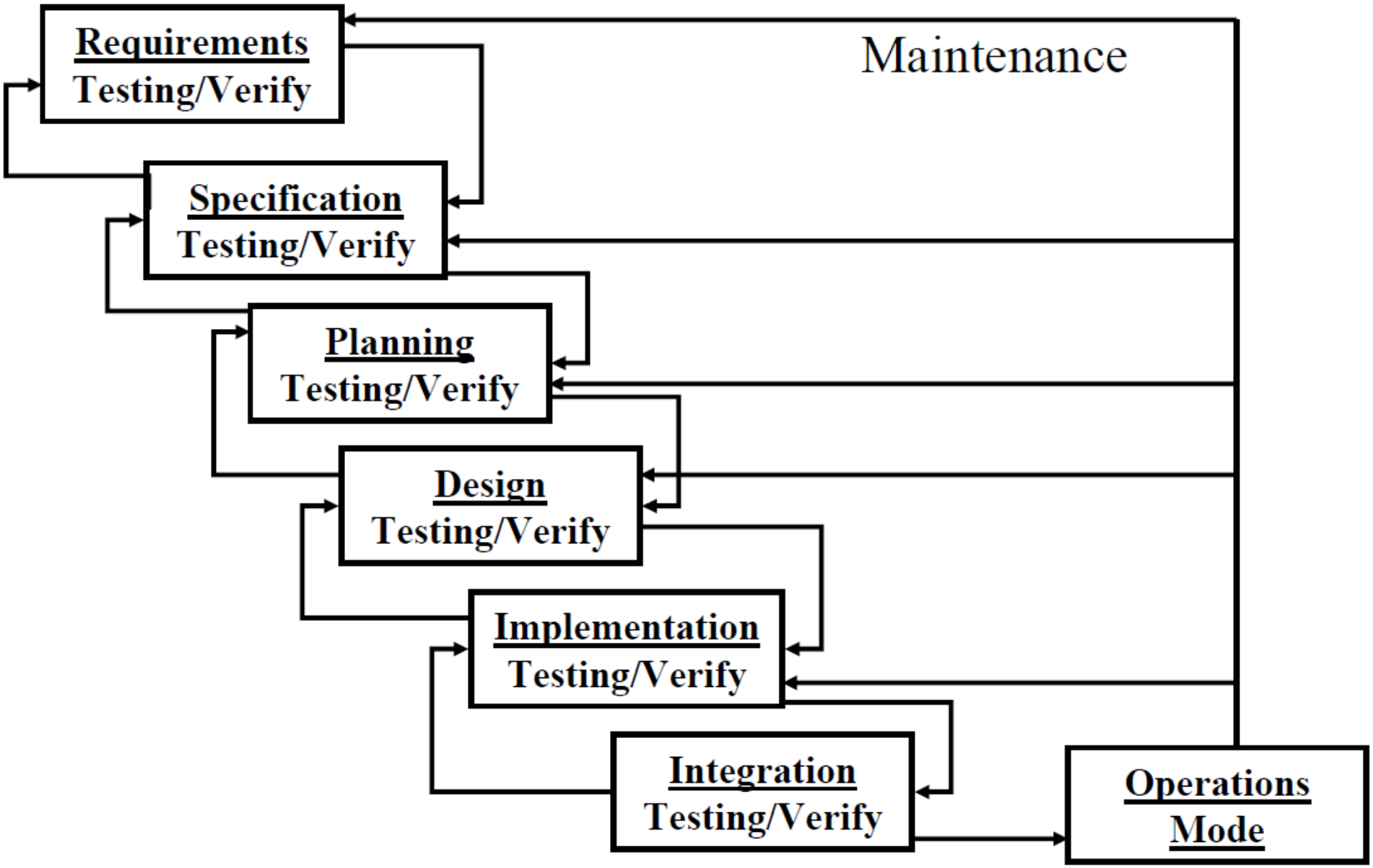


A Waterfall Process Model [Sommerville]



Each stage could span for months or years!

Waterfall Variation



Software Development Activities

- Note: The exact terminology varies somewhat

- We'll try to follow those used in Sommerville

Requirements Capture

- What are the customer needs?
  - Features
  - Usability
  - Reliability/Quality
  - Performance
- What shall we build to fulfill those needs (sometimes called a *specification*)?
- Usually results in a requirements document/list

- How can you determine if the requirements are correct?

Design

- Identify and describe the software system abstractions and their relations
  - Software Architecture (e.g. client/server, layered, etc.)
  - Software design
  - Database design
  - Interface design
  - Reusable (e.g. open-source) component selection
    - Licensing issues

Implementation

- Programming and debugging based on the design and specifications

- Traditionally an individual activity with no standard process
  - Agile challenges that tradition

Testing

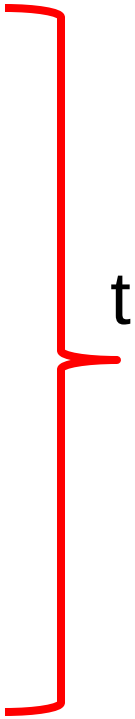
- *testing* can be considered as a legacy term

- We will often use the term *defect removal* because modern teams use a variety of defect removal methods beyond testing alone:
  - [examples…?]

Testing

- *testing* can be considered as a legacy term

- We will often use the term *defect removal* because modern teams use a variety of defect removal methods beyond testing alone:
  - Pair Programming
  - Test-Driven Development
  - Unit-Level Testing
  - Static Analysis
  - Code Reviews
  - Integration and Regression Testing
  - System-Level Testing

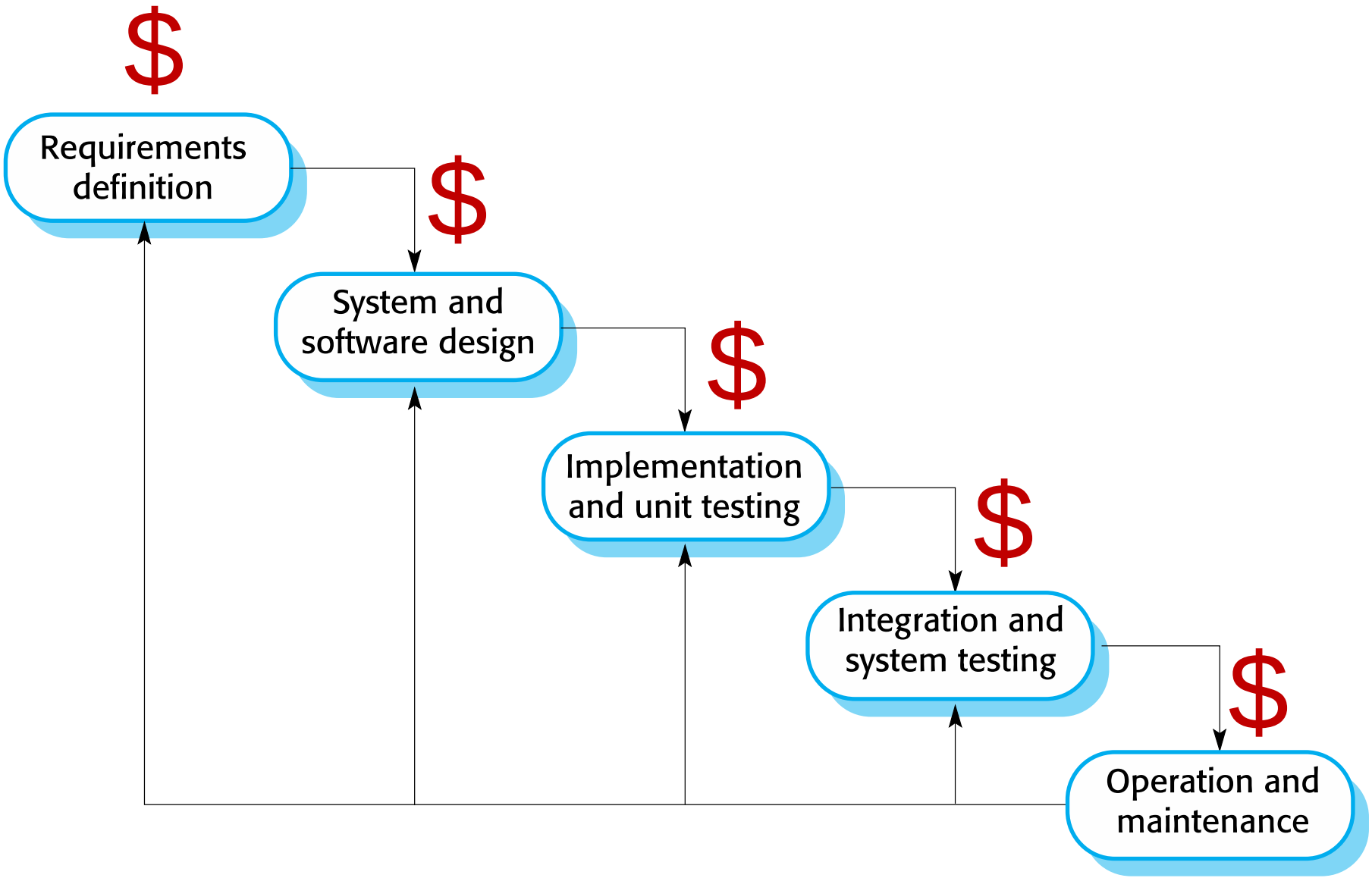


We'll cover these later in the semester

More About Testing

- Defects are vastly cheaper to remove early in the project

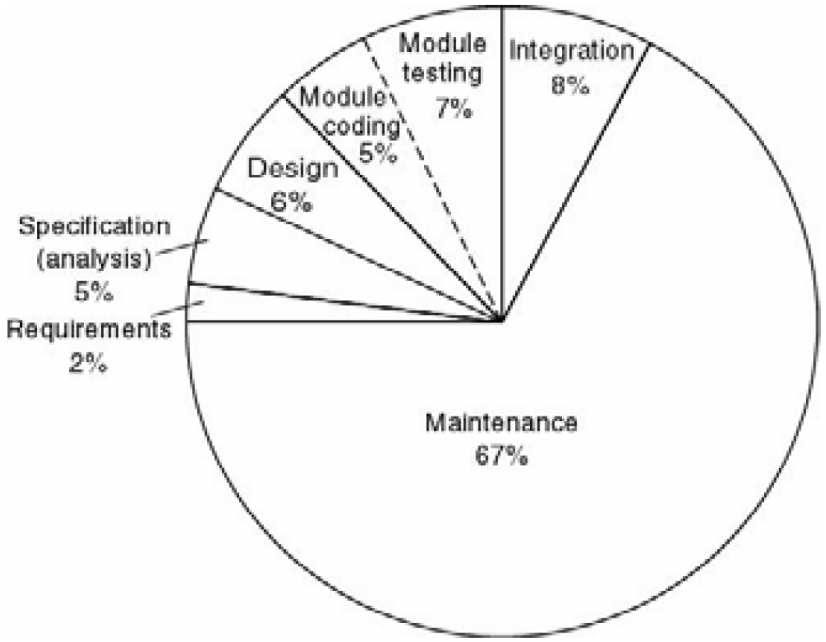
Approximate Relative Cost of Each Phase



Approximate Relative Cost of Each Phase

- 1976–1981 data

- Maintenance constitutes 67% of total cost

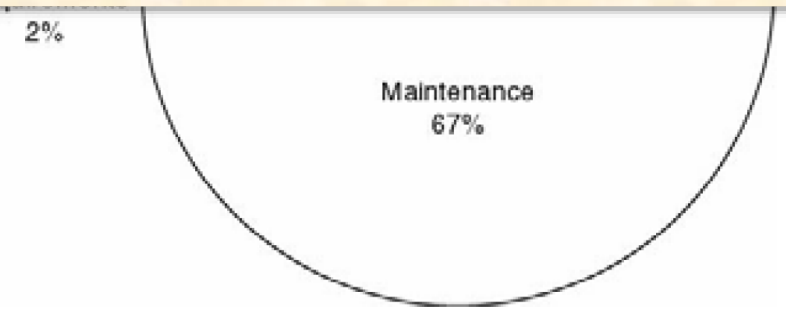


Approximate Relative Cost of Each Phase

- 1976–1981 data

Up to **90%** of software cost is spent on **maintenance**

[Erlikh'00]



Empirical data based on the waterfall model

- Maintenance activities divided into four classes*:
  - Adaptive – changes in the software environment (about 20% of all changes)
  - Perfective – new user requirements (20%)
  - Corrective – fixing errors, bugs (20%)
  - Preventive – prevent problems in the future.

*Lientz and Swanson Survey

Empirical data based on the waterfall model

- 60 to 70% of faults are specification and design faults

- Data of Kelly, Sherif, and Hops [1992]
  - 1.9 faults per page of specification
  - 0.9 faults per page of design
  - 0.3 faults per page of code

Waterfall Applications

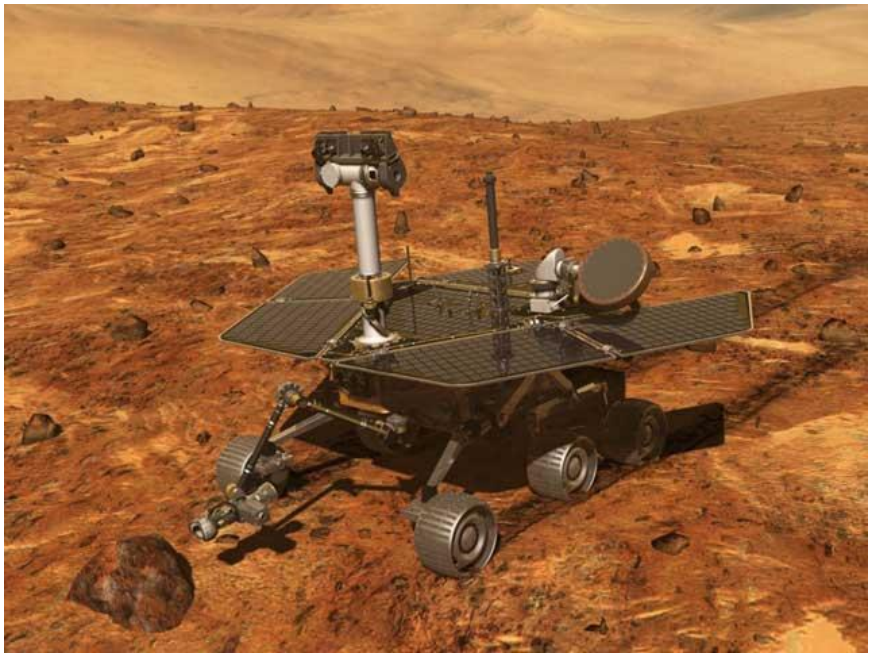
- Works best with stable requirements and technologies
- Not a bad choice for routine IT-like projects
  - e.g., payroll application for company B, which is similar to the payroll application delivered in the past for company A
- Arguably useful in any project benefitting from up-front plans

Waterfall Applications

- Arguably the best choice for contractual development
  - Executives in suits and their attorneys gather in a conference room
  - Contracts are signed, specifying:
    - what will be built
    - when it will be completed
    - how much it will cost and
    - penalties for changes

Waterfall Applications

- Arguably the best choice for contractual development
  - Executives in suits and their attorneys gather in a conference room
  - Contracts are signed, specifying:
    - what will be built
    - when it will be completed
    - how much it will cost and
    - penalties for changes
  - Widely used in
    - government
    - aerospace and some
    - enterprise IT



Waterfall Disadvantages and Practical Issues

- Changes waste the painfully created up-front planning
- Inflexible partitioning of development into gated stages
  - may idle resources (e.g., next stage cannot start before current one)
- And, if not used, attempts to achieve the benefits of an agile process without the activities required to be agile

Waterfall Disadvantages and Practical Issues (contd.)

- Big-Bang Integration, if actually used, leads to chaos (Imagine… if we each independently created a component of a car and then try to put everything together)

- Invisible problems (e.g., we built the wrong product) may lead to complete failure

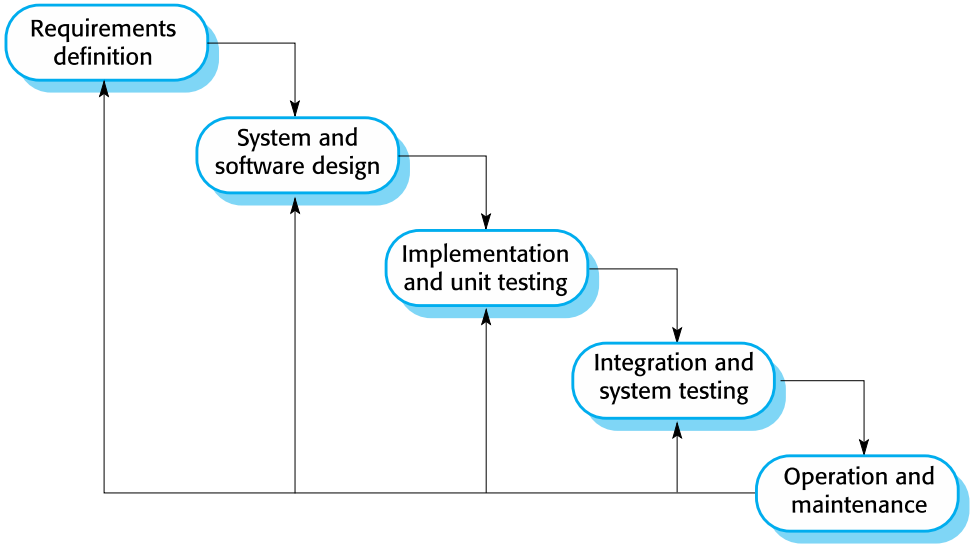
Waterfall still exists…

- still a standard
- software engineering textbooks still based on it
- many managers still adhere to it
- rarely followed by the programmers

Waterfall Summary

- Strongly emphasizes up-front planning — *Big Design Up-Front* (BDUF)
- Some phases produce only documents (e.g., requirements)

- Typical sequential phases:
  - Requirements
  - Design
  - Implementation
  - Testing
  - Maintenance



Waterfall Summary

- Gated — Complete current stage before beginning the next
  - Each phase built on the planning of a previous phase
- May not produce any code until everything is planned in detail
- Builds the product with "Big bang" integration of code modules

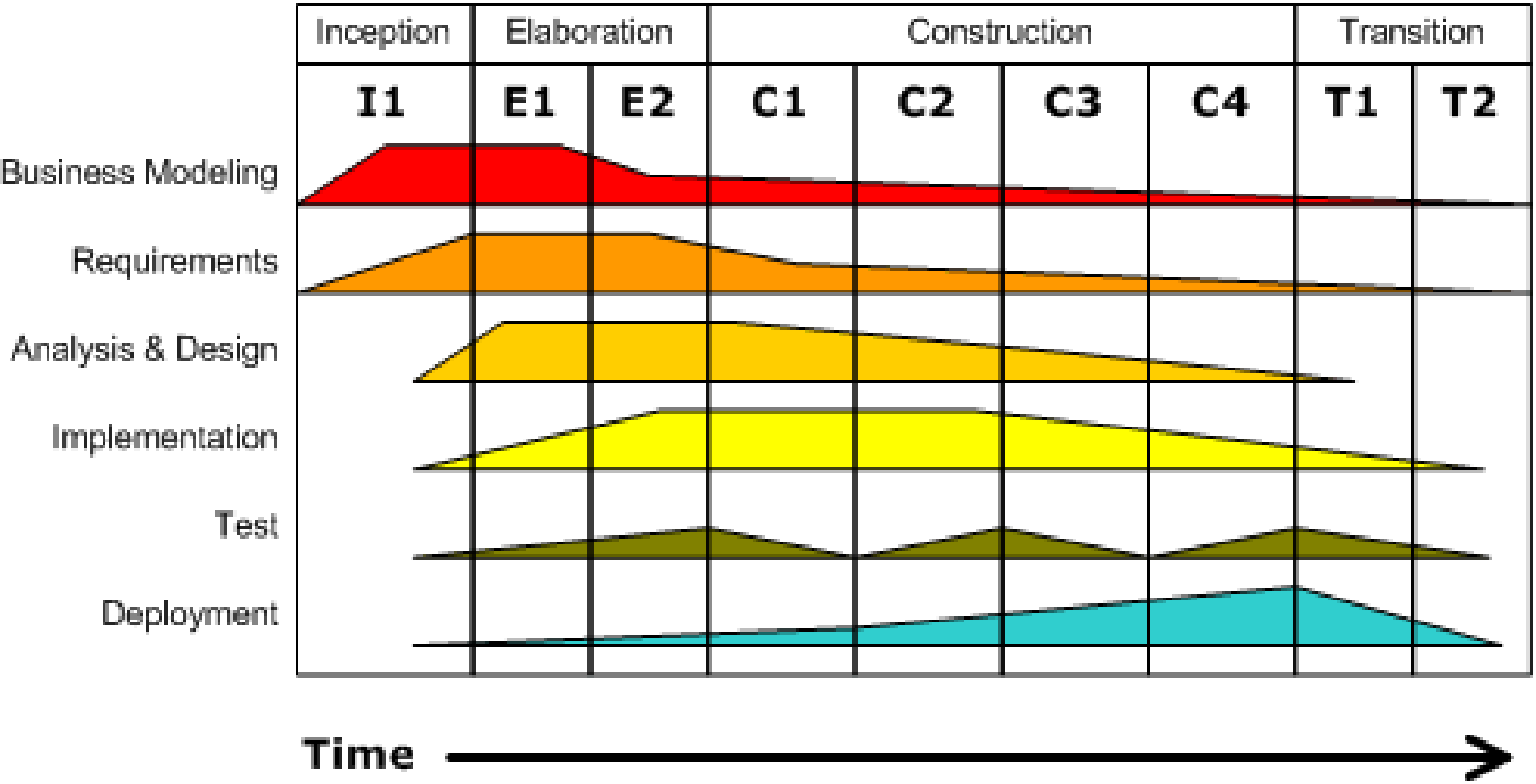
Adaptations of Waterfall and Incremental Development (not covered in 471)

- Prototyping Model
- Rapid Application Development
- Evolutionary Process Models
- **Spiral Model**
- Component Assembly Model
- Concurrent Development Model
- Formal Methods Model
- **Unified Process**

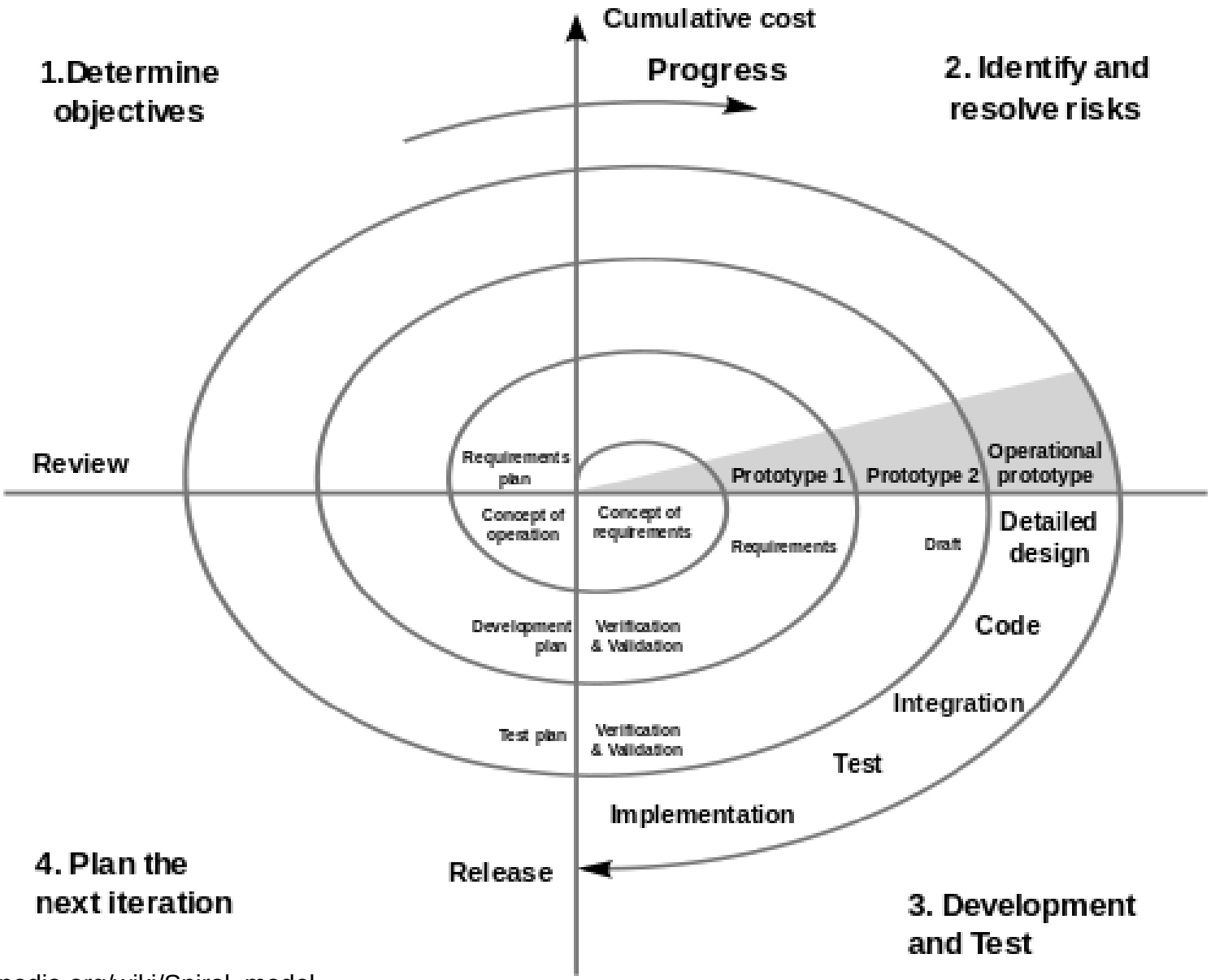
Unified Process

Iterative Development

Business value is delivered incrementally in time-boxed cross-discipline iterations.



Spiral model



https://en.wikipedia.org/wiki/Spiral_model

How Would a Team Write *War and Peace*?

How Would a Team Write *War and Peace*?

- English translations contain ≈ 560,000 words
- Tolstoy had 10 years to write *War and Peace* solo

How Would a Team Write *War and Peace*?

- English translations contain ≈ 560,000 words
- Tolstoy had 10 years to write *War and Peace* solo

- 13 Software Engineers routinely develop:
  - 115 KLOC (≈ 560,000 "words" ≈ *War and Peace*) in 14 months

How Would a Team Write *War and Peace*?

- English translations contain ≈ 560,000 words
- Tolstoy had 10 years to write *War and Peace* solo

- 13 Software Engineers routinely develop:
  - 115 KLOC (≈ 560,000 "words" ≈ *War and Peace*) in 14 months
  - How many LOC will an engineer write on average per workday?
    - (Assume each month has 20 workdays)
    - (whiteboard only, in-class solution)