Can we mitigate the essential problems of software?

- Ada and other high-level language advances
- Graphical programming
- Object-oriented programming
- Artificial intelligence
- Program verification
- Expert/Recommender systems
- Environments and tools
- "Automatic" programming

New Trends in Software Engineering

- Software evolution
- Agile processes
- Product lines
- Service oriented software
- Software visualization
- Improving OO
- Generative programming
- Empirical software engineering

What to Expect in CS471?

What to Expect in CS471: "Question all the Answers"

What to Expect in CS471: "Question all the Answers"

- Software Engineering is an active area of research
- Best practices continue to emerge from this research
- We supplement our texts with selected research papers

What to Expect in CS471: "Question all the Answers"

- You won't leave CS471 with all the answers
- You will leave thinking critically about the answers!
- You will leave on the trail of continuous education!

CS471: Related Courses

- CS472 provides a deep dive into software design
- CS474 provides a deep dive into software quality
- CS481 (Fall'18) provides a deep dive into a real world software project
- Continuing education following graduation!

HW1 Background Information

Issue Tracking Systems

Bugzilla

GitHub

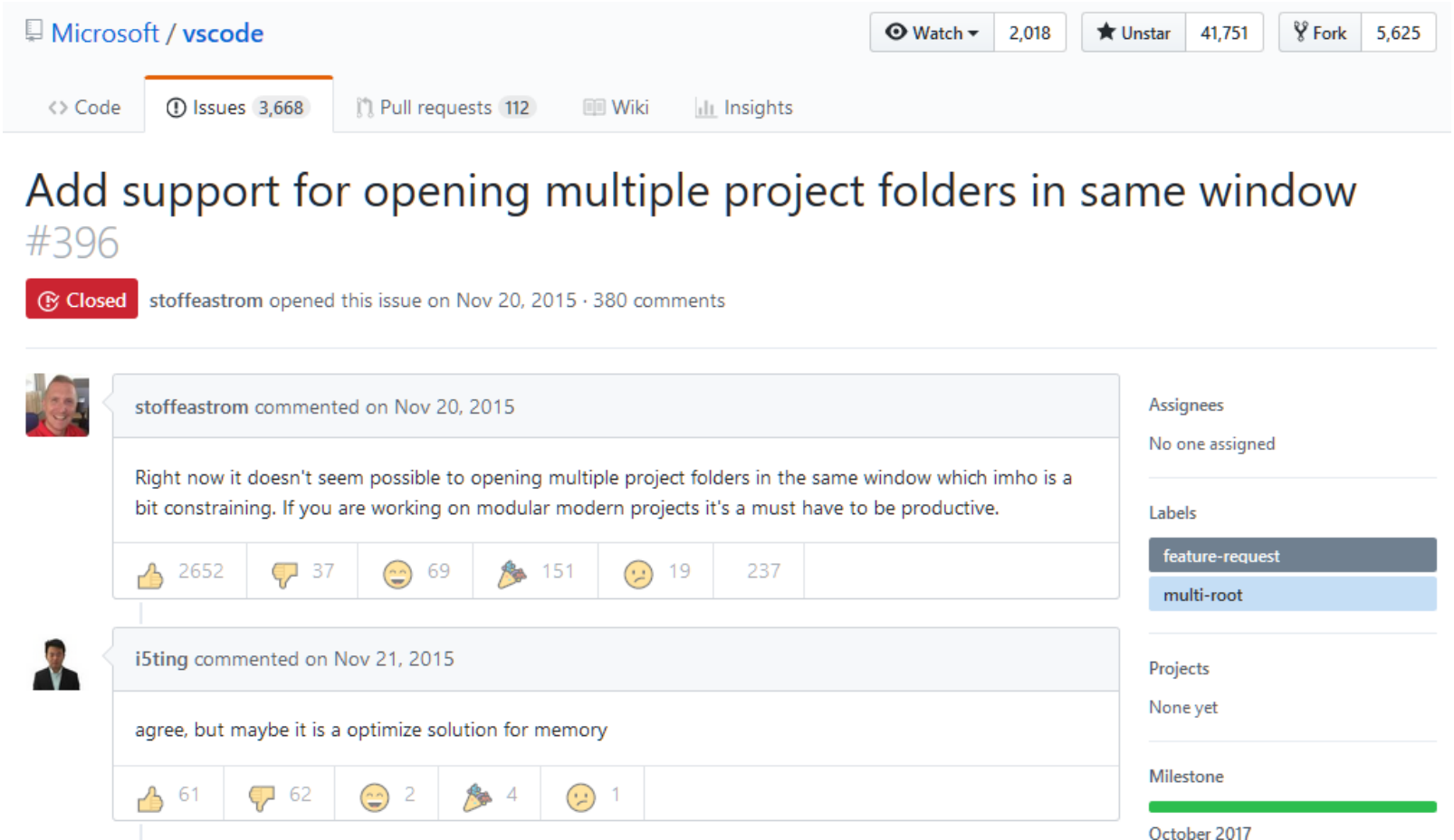
Atlassian JIRA

trac
Integrated SCM & Project Management

- Contain "bug/defect" reports and feature requests

Feature Request

- Describes what new functionality the product should implement
- https://github.com/Microsoft/vscode/issues/396



Bug/Defect Report

- Describes something the product has not correctly implemented

- When writing a bug report, what information should you provide?

- https://developer.mozilla.org/en-US/docs/Mozilla/QA/Bug_writing_guidelines

Defect Report Template for Class Project

Short descriptive title:

Description

Steps to Reproduce:

1. TBD
2.
3.

Actual Results:

TBD

Expected Results:

TBD

Other notes:

TBD

Defect Report Template for Class Project

Short descriptive title:

Description

Steps to Reproduce:

1. TBD
2.
3.

Actual Results:

TBD

Expected Results:

TBD

Other notes:

TBD

Additional information that can be useful:
- environment (OS, platform, version, etc.)
- logs
- memory dumps
- stack traces, etc.

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)

Waterfall vs. Agile

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

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

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*

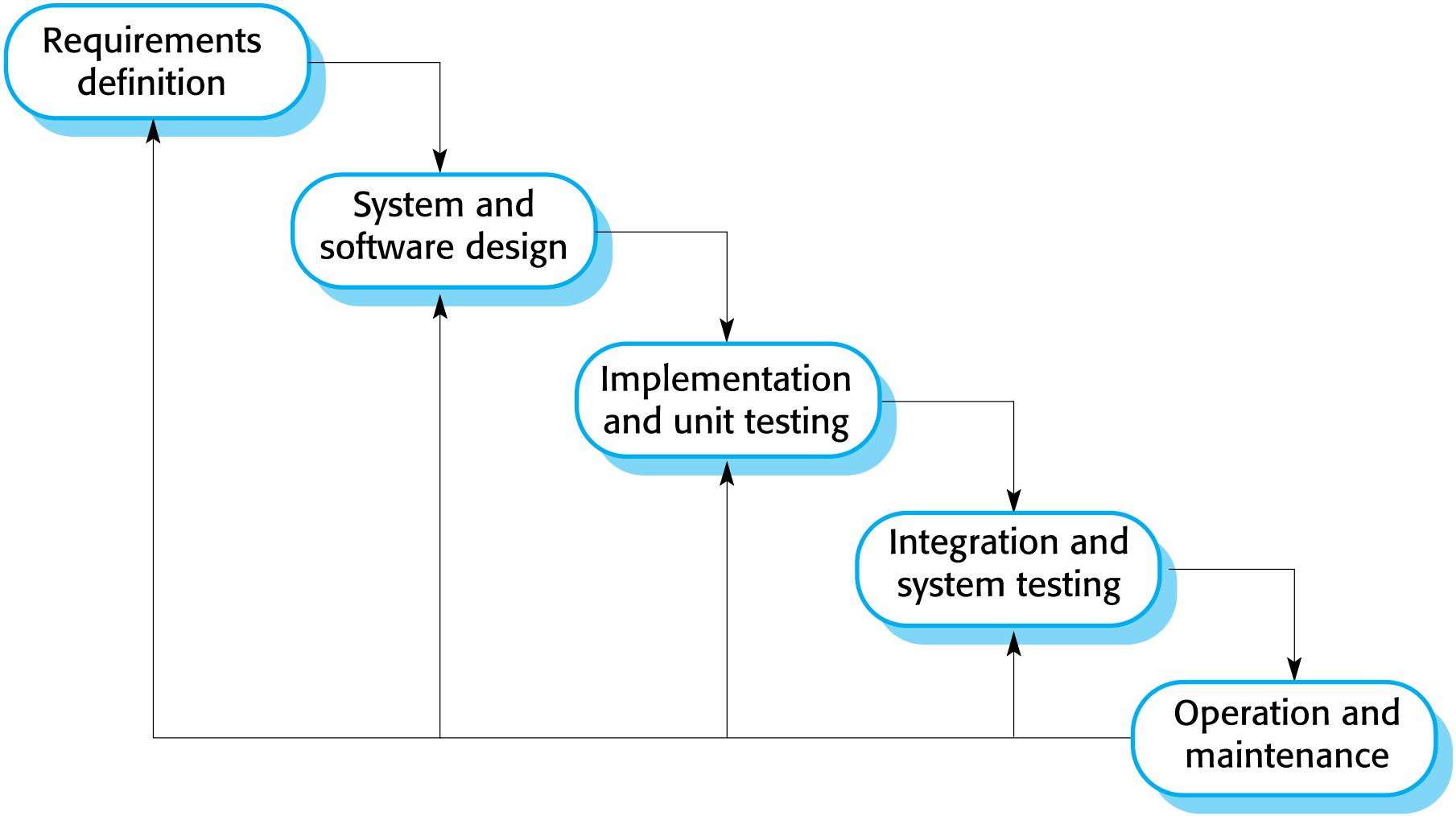
- Your mileage may vary!

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

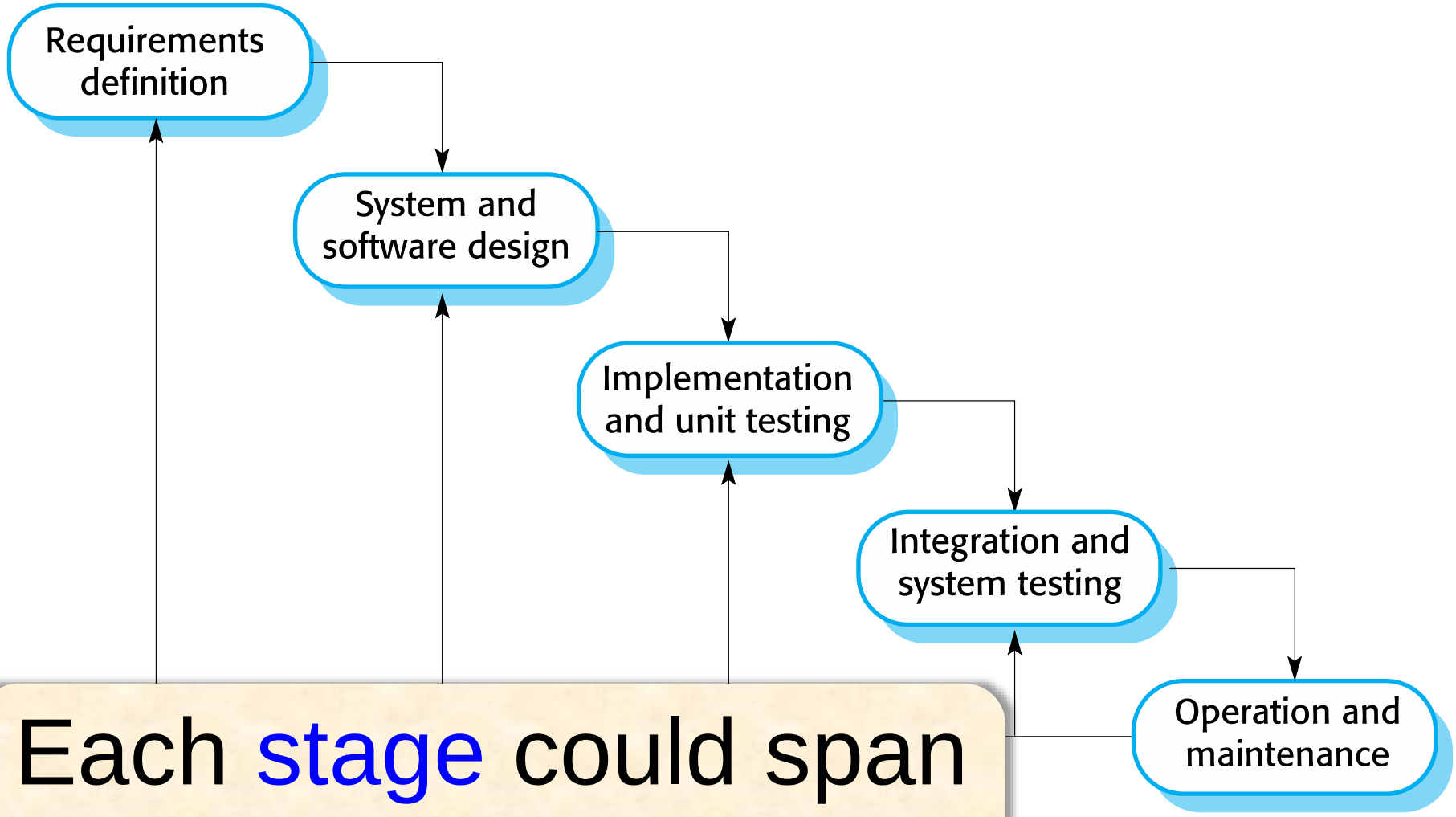
- Prototyping Model
- Rapid Application Development
- Evolutionary Process Models
- Spiral Model
- Component Assembly Model
- Concurrent Development Model
- Formal Methods Model
- Unified Process

The Waterfall Process Model

A Waterfall Process Model [Sommerville]

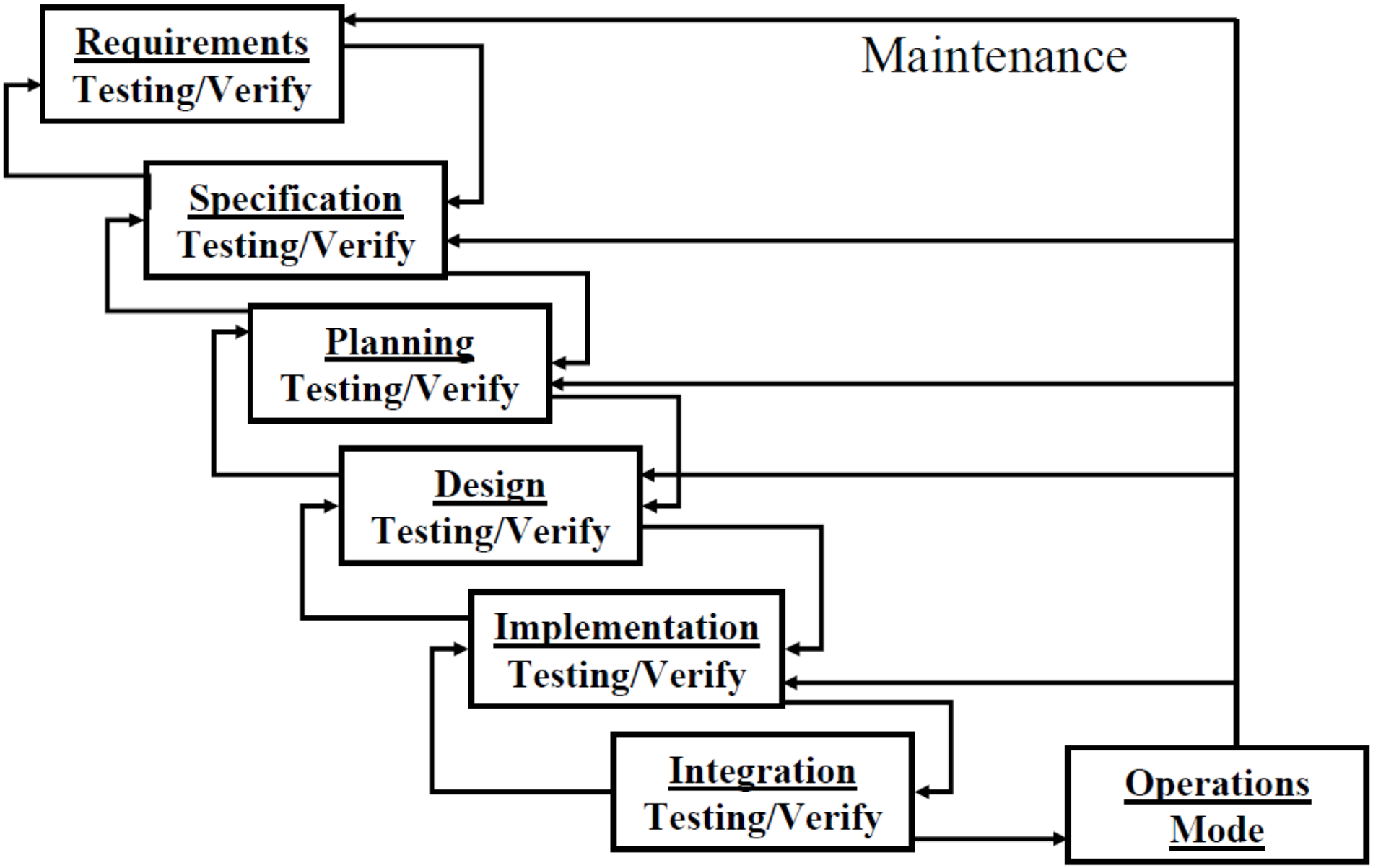


A Waterfall Process Model [Sommerville]



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

- The question that should be asked is …

Requirements Capture

- What does the customer need?
  - Features
  - Usability
  - Reliability/Quality
  - Performance
- What shall we build to fulfill those needs (sometimes called a *specification*)?
- Usually results in a requirements document/list
- Question: How can you determine if the requirements are correct?

Design

- The question that should be asked is …

Design

- How will the software work?
  - 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
- 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:
  - TBD…

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 …

More About Testing

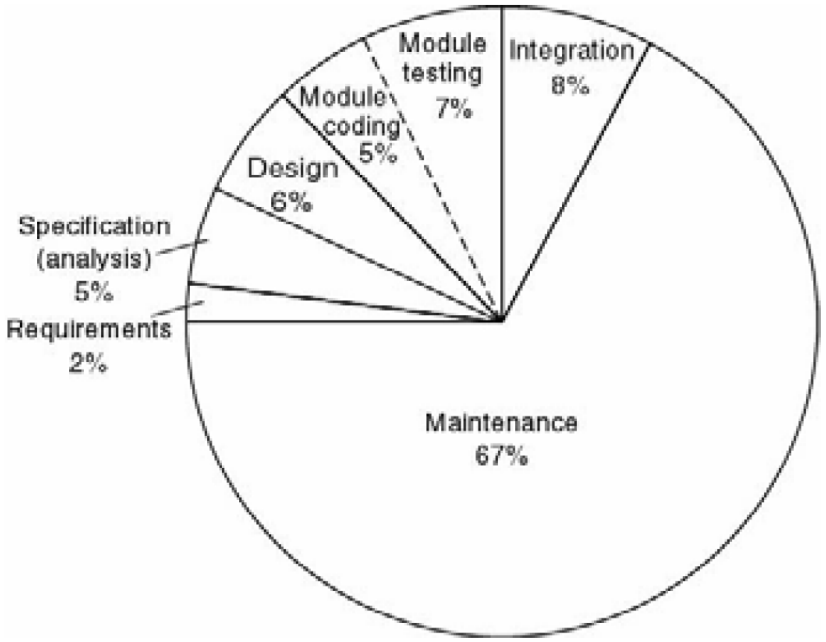
- Defects are vastly cheaper to remove early in the project

Approximate Relative Cost of Each Phase

Approximate Relative Cost of Each Phase

- 1976–1981 data

- Maintenance constitutes 67% of total cost

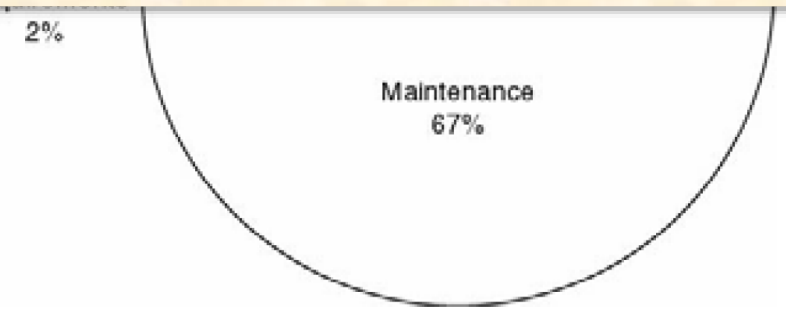


Approximate Relative Cost of Each Phase

- 1976–1981 data

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

[Erlikh'00]



Empirical data based on the waterfall model

- Survey by Lientz and Swanson: 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 (20%)
  - Preventive – prevent problems in the future.

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 Characteristics

Waterfall Characteristics

- Activities performed in sequential *stages*
- Gated — Complete current state before beginning the next
- Note heavy up-front planning — *Big Design Up-Front* (BDUF)
- Big Bang Integration