

**22C:180/55:180**  
**Software Engineering--**  
Architecture & Design of Software Systems

- **Instructor:**
  - Prof. Jon Kuhl, University of Iowa, Department of Electrical and Computer Eng.
    - Office: 305S2 CC
    - Phone: (319) 335-5958
    - E-mail: kuhl@eng.uiowa.edu
    - Office Hours: t.b.d

**22C:180/55:180**  
**Software Engineering--**  
Architecture & Design of Software Systems

- **Teaching Assistants:**
  - Bob Karl,
    - Office: 108 CC #13
    - E-mail: rakarl@cs.uiowa.edu
    - Office hours: t.b.d
  - Tammy Scherbring
    - Office: t.b.d.
    - E-mail: tscherbr@cs.uiowa.edu
    - Office hours: t.b.d.

22C:180/55:180  
Software Engineering--  
Architecture & Design of Software Systems

- Administrative Trivia:
  - Class web site:  
<http://www.icaen.uiowa.edu/~softeng>
  - Text (available at IMU Bookstore):
    - C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*, Prentice Hall PTR, 1998. ISBN: 0-13-748880-7.

22C:180/55:180  
Software Engineering--  
Architecture & Design of Software Systems

- Required Background:
  - Senior or graduate status in CS, ECE, or other computer-related discipline.
  - Programming experience.
  - Prior exposure to an object-oriented programming language such as C++ or JAVA.

**22C:180/55:180**  
**Software Engineering--**  
Architecture & Design of Software Systems

- **Course Objectives:**
  - Provide an introduction to modern software development practices and technologies
  - Motivate software system development from an “architectural” perspective.
  - Introduce practical software design skills.
  - Expose students to real-world examples and state-of-the-art industrial practices.

22C:180/55:180 Software Engineering

5

**22C:180/55:180**  
**Software Engineering--**  
Architecture & Design of Software Systems

- **Assignments, Exams, Projects:**
  - Three exams.
    - Exams count 50% of semester grade.
  - Homework/Projects
    - Individual homework assignments
    - Group project during second half of semester.
    - Homework/projects count 50% of semester grade.

22C:180/55:180 Software Engineering

6

22C:180/55:180  
Software Engineering--  
Architecture & Design of Software Systems

- What's Different About This Course
  - Focus on software architecture
  - Orientation toward pragmatic OOA/OOD techniques
    - UML
    - Design patterns

## Software Engineering

- Broad definition: The methods, processes, and technologies via which software systems are developed and maintained.
- Encompasses many areas:
  - Planning and managing software projects
  - Requirements Analysis and Specification
  - System Modeling and Design
  - System Implementation
  - Testing and quality assurance
  - Maintenance

# Software Engineering

- Why is it important:
  - A simple analogy:
    - Programming skills are to software development as carpentry skills are to building construction.
  - Software systems can be enormously complex.
  - Programming skills, knowledge of algorithms and data structures, etc., don't provide the tools to specify, design, and maintain complex software systems.

# Complexity of Software Systems

- Logical vs. Physical complexity:
  - Physical complexity--How hard is it to implement the system
  - Logical complexity--How difficult is it to comprehend the operation of the system--e.g. how many states does it have.
  - Even modest-sized software systems can have overwhelming logical complexity.

## Complexity of Software Systems

- The Dangerous Fallacies of Software:
  - Software has low physical complexity, therefore it is “easy” to build software systems.
  - Building software systems is “just a matter of programming”.
  - Software systems can be built quickly and inexpensively, compared to hardware systems.

## The Reality of Software

- History of commercial software development is fraught with horror stories.
  - Numerous high-profile development efforts have “gone down in flames.”
  - faulty software systems have killed people.
- Even today, large software projects routinely miss cost and delivery schedules by large margins.

## Current State of The Software Industry

- Intense competition demands increasingly short and cost-effective development cycles.
- Products require frequent upgrade to include new features.
- Customers require (or demand) higher levels of reliability.
- Legacy systems continue to be a major factor.

22C:180/55:180 Software Engineering

13

## Some Elements of a Software Engineer's Repertoire

- Ability to understand customer's requirements and expectations
- Ability to translate these into precise specifications
- Ability to plan a system development effort
  - establish timelines
  - allocate resources
- Ability to perform conceptual modeling and high-level system design

22C:180/55:180 Software Engineering

14

## A Software Engineer's Repertoire (continued)

- Ability to evaluate design alternatives and tradeoffs.
- Ability to carry-out detailed system modeling and design
- Ability to implement a system based upon the system design
- Ability to plan and conduct validation and test activities.

## A Software Engineer's Repertoire (continued)

- Ability to work in team environment
- Understanding of product-family engineering, component-based design principles, and software reuse
- Understanding of software maintenance issues
- Oh yes, and he/she must also be a proficient programmer.

## Software Architecture

- “*Good heavens! For more than forty years I have been speaking prose without knowing it.*”  
Moliere
- Software Architecture is concerned with the system-level structure of a software system.
- Describes basic system elements, interactions among these elements, and patterns that guide their composition.

## Software Engineering Fundamentals

- The Software Development *Process*:
  - The framework for the set of tasks that are required to develop a software system.
  - Process defines *how* a software product is developed and maintained.
  - A well-defined and rigorously enforced process forms the basis for high-quality software development.

## Process Maturity

- Refers to the effectiveness and comprehensiveness of an organization's software engineering practices:
- The Software Engineering Institute (SEI) has developed a *Capability Maturity Model* (CMM) to assess process maturity.
- The CMM defines five levels of process maturity and has an associated assessment process to determine an organization's level.

## The SEI CMM Levels

- **Level 1: Initial**
  - Ad hoc
  - Occasionally chaotic
  - Few, if any, defined processes
  - Project success primarily dependent on individual talent and initiative.
- **Level 2: Repeatable**
  - Basic project management processes to track cost and schedule.
  - Enough process discipline to repeat earlier success on previous projects of a similar nature.

## The SEI CMM Levels

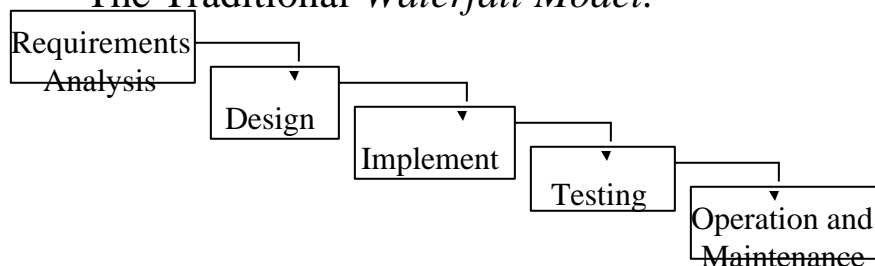
- Level 3: Defined
  - Documented and standardized software process for both engineering and management functions well integrated throughout the organization.
  - All projects utilize approved version of this process
  - Ongoing efforts to improve process
- Level 4: Managed
  - Quantitative quality and productivity goals for each project
  - Detailed measures of process and product quality routinely collected.
  - Process and products are quantitatively understood and controlled using collected measures.
- Level 5: Optimizing
  - Continuous process improvement via quantitative feedback from the process

22C:180/55:180 Software Engineering

21

## Standard Software Life-cycle Models

- The Traditional *Waterfall Model*:

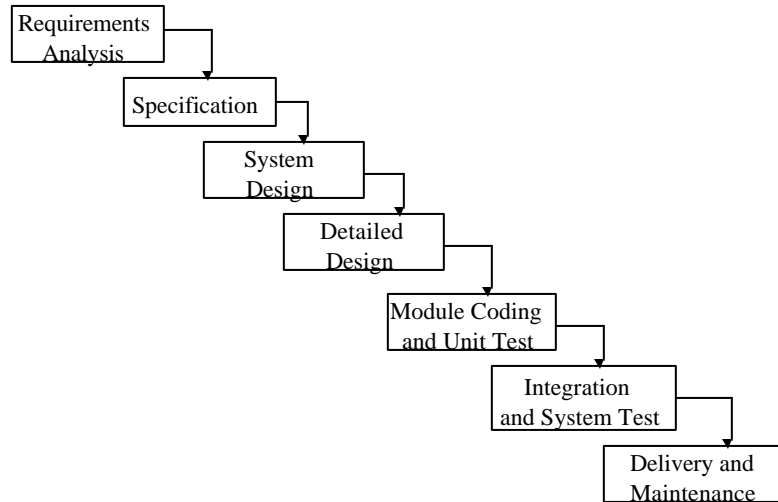


22C:180/55:180 Software Engineering

22

## Standard Life-cycle Models

- A more detailed waterfall model:



22C:180/55:180 Software Engineering

23

## The Waterfall Model

- Requirements Analysis and Specification:
  - Understand customer's requirements (system behavior)
  - Assess feasibility
  - Carefully document requirements to satisfaction of customer
    - conceptual model
    - system functionality (functional requirements)
    - system attributes (non-functional requirements)
    - hardware requirements
    - maintenance requirements

22C:180/55:180 Software Engineering

24

## The Waterfall Model

- Design Stage:
  - System (Architectural) Design:
    - Determine system-level structure
    - Identify key system elements and interactions
    - Allocate system functions
  - Detailed Design
    - Identify system modules (objects), allocate functionality to these. Define module (object) behavior at algorithmic and data-structure level.

## Waterfall Model

- Module Coding and Test
  - Transform module designs into code
  - perform unit testing of modules
- System Integration and Test
  - Put modules together to form functioning system
  - Test and validate system behavior/performance.

## Waterfall Model

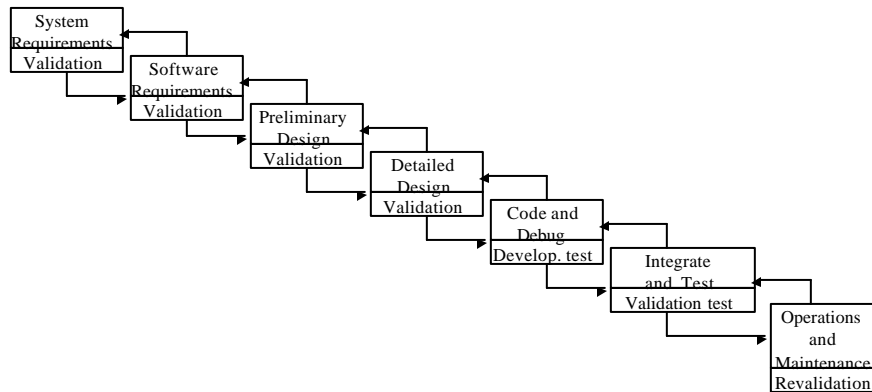
- Delivery and Maintenance:
  - Deliver product to customer with manuals and documentation
  - Train customer to use system
  - Service customer bug reports.

## Waterfall Model

- Limitations:
  - Highly simplified view of typical software life-cycle
  - Treats software development as a “manufacturing” process rather than a “problem-solving” process.
  - In practice, there are numerous feedback loops among stages.
  - Doesn’t deal with system upgrade/evolution.
  - Doesn’t provide for incremental development, rapid prototyping, etc.

## Other Life-cycle Models

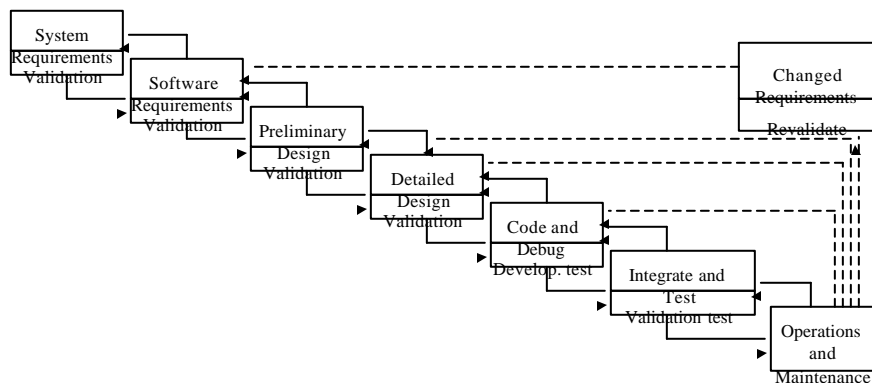
- Software life-cycle Model with Feedback:



22C:180/55:180 Software Engineering

29

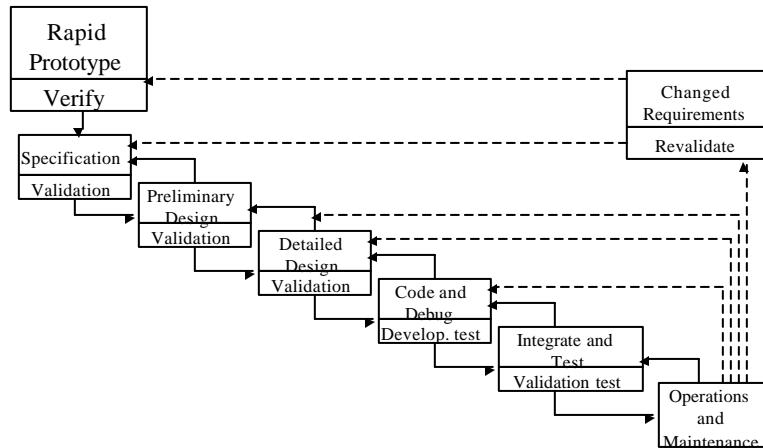
## A Lifecycle Model Supporting Maintenance and Upgrade



22C:180/55:180 Software Engineering

30

## Rapid Prototyping Model

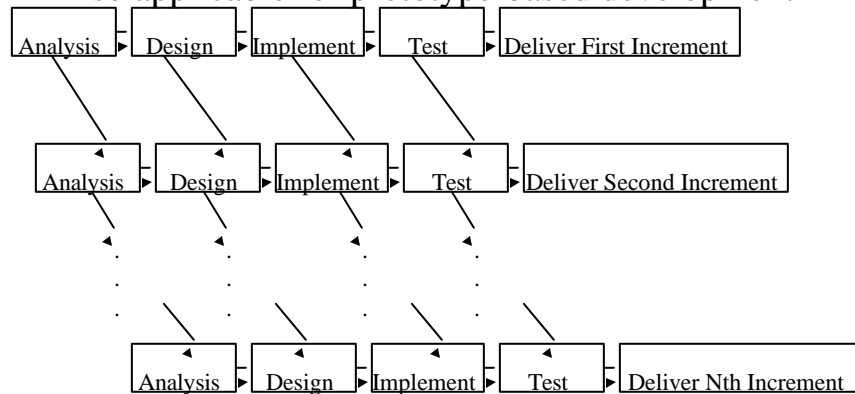


22C:180/55:180 Software Engineering

31

## An Incremental Life-Cycle Model

- Also applicable for prototype-based development

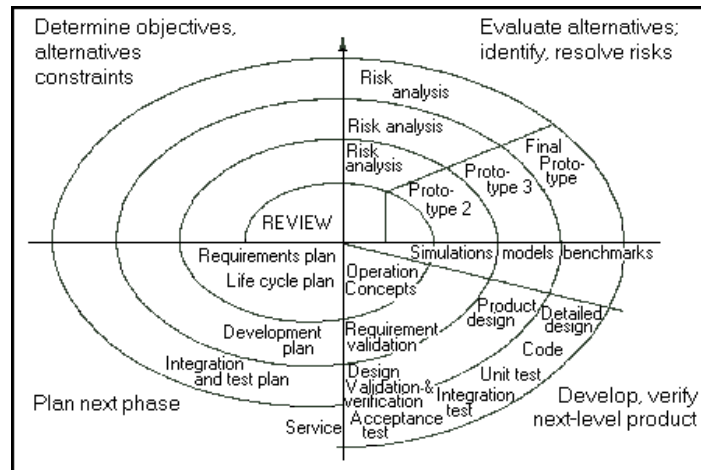


22C:180/55:180 Software Engineering

32

## Boehm's Spiral Life-Cycle Model

- From Boehm (1988) p. 434



## Boehm's Spiral Life Cycle Model

- Each Clockwise loop (360 degrees) starting at -x axis represents one development phase.
- Each development phase is comprised of four major activities:
  - Objective setting
  - Risk Assessment and reduction
  - Development and validation
  - Planning for next phase

## Boehm's Spiral Model

- More detailed list of steps in each phase
  - Elaborate product & process objectives and constraints
  - Elaborate alternative product & process solutions
  - Evaluate alternatives relative to objectives
  - Carry out risk reduction by prototyping, simulation, benchmarking, etc.
  - Identify and refine the most attractive alternative
  - Enter an appropriate process to develop the alternative
  - Validate and prepare plans for next cycle
  - Final review