# Software Reliability and Security

## Module 3

## Fall 2017

# Presentation/Lecture Schedule and Report Due Dates

- Presentation 1
  - Related background paper
  - Jan 27, Feb 1, 3
- Presentation 2
  - Project proposal
  - March 1, 3, 8
- Presentation 3
  - Final project report
  - March 24, 29, 31

- Lectures
  Jan 13, 18, 20, 25, 27
  Feb 1, 3, 8, 10, 15, 17
  March 1, 3, 8, 10, 15, 17, 22, 24, 29, 31

- Project Proposal Due
  Tuesday, February 28

- Final Project Report Due
  Monday, April 10

- Final Exam
  Wednesday, April 12, 10:00am

# Outline

- Software Quality
- Software Process Models
- Methods for Reliable Software

# Software Quality

- Software quality [Voas 2002] requirements can be specified based on various attributes
- The composition of some or all of the non-functional attributes ("ilities")
  - Reliability (R), Performance (P), Fault Tolerance (F), Safety (Sa), Security (Se), Availability (A), Testability (T), Maintainability (M)
  - Many other "ilities"

# Some Other Quality Attributes

- Usually Quantitative
  - Correctness –generate outputs that match specification
  - Capability – provides all the required functionalities
  - Resilience - provides outputs under unexpected circumstances
  - …
- Usually Subjective (User Satisfaction)
  - Usability - easy to use
  - Installability – easy and fast to install
  - Documentation – easy to understand documentation

# Quality Equation

- The Software Quality (Q) equation
  - Q = aR+bP+cF+dSa+eSe+fA+gT+hM
  - Example: Q = 3R+2P+5F+3Sa+5Se+4A+3T+2M
- More than one tradeoffs between the 'ilities"
  - Example: c may increase if b decreases, …
- Total system quality is complicated when a number of components connected with different priorities of "ilities"
  - Q(C1) = aR+bP+cF+dSa+eSe+f4+gT+hM
  - Q(C2) = iR+jP+kF+lSa+mSe+n4+oT+pM

# Questions on Quality Equation

- What is Q?
  - Integer
  - Floating point value
  - Probability
  - A tuple as (2, 5, 3, 2, 5, 4, 3, 5)
  - Color as yellow, blue, …
- The coefficient may change even if the software itself does not change – software rot and software quality changes
  - For example: security coefficient "c" may change if a new attack surfaces

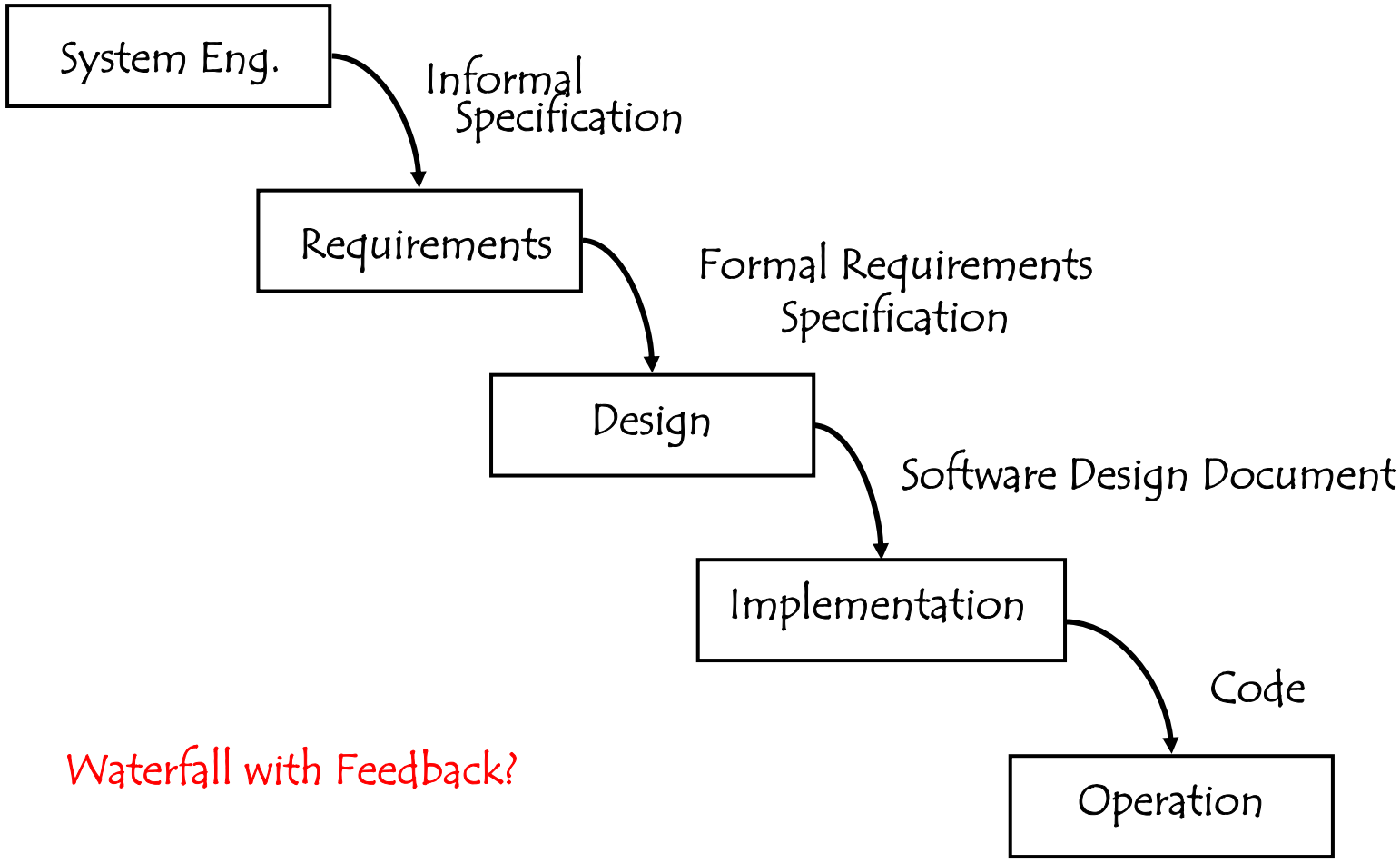
# Software Process Models

- Software engineering – from various sources
  - Software development is not only programming
  - Multi-person construction of multi-version software
  - Engineering techniques and methods for building large software systems by a number of people in an systematic way
  - Each software process model includes a set of steps to build a software product - software life cycle model

# Software Process Models

- Most software life cycle models include the following steps
  - Requirements
  - Specification
  - Design
  - Programming
  - Integration
  - Testing (may be attached to any steps?)
  - Operation and Maintenance
- Some most commonly used models
  - Waterfall Model
  - Prototyping model
  - Spiral model

# Waterfall Model

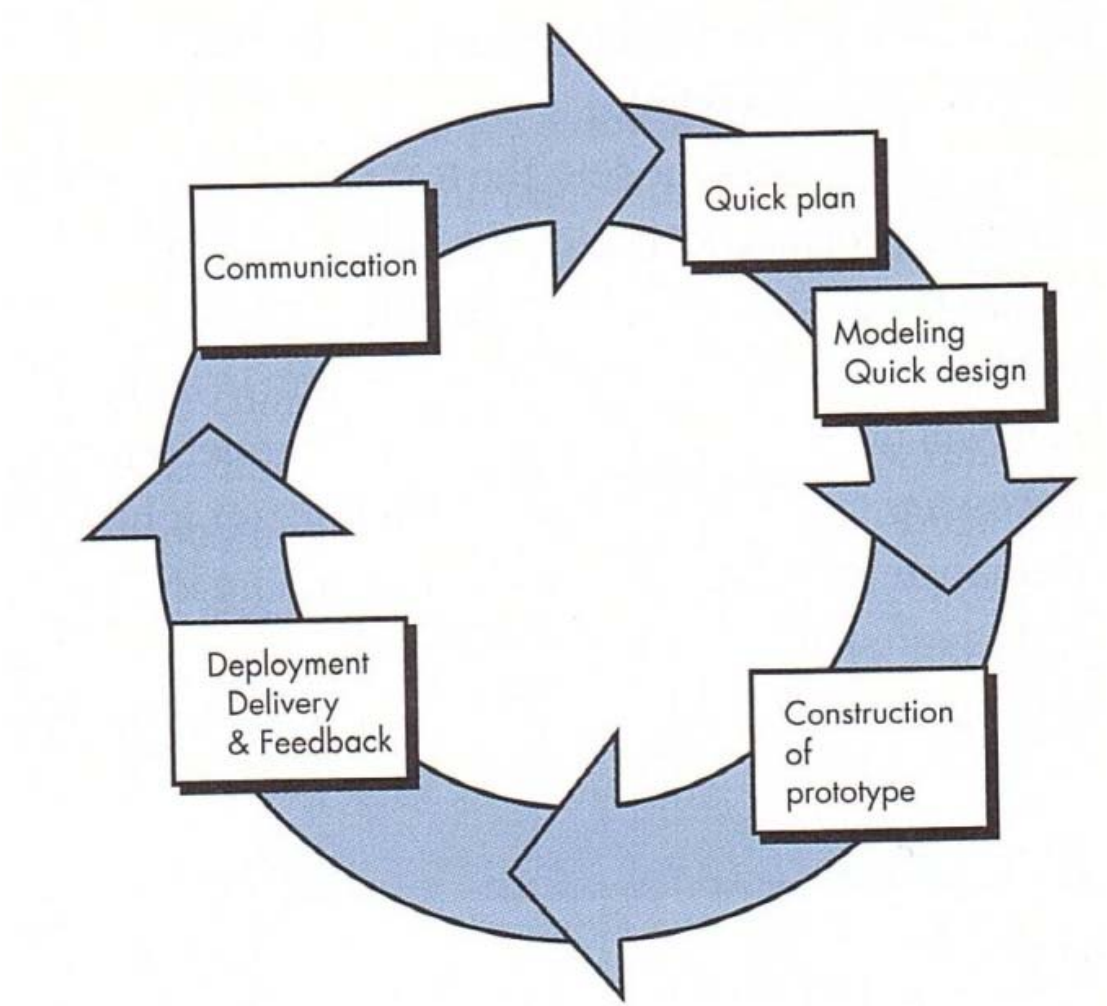



Waterfall with Feedback?

# Waterfall – Advantages and Disadvantages

- Advantages
  - Prescribes a strict disciplined approach following well-defined tasks
  - Separation of phases and transitions among them – separation of tasks
  - Documentation helps reduce maintenance
- Disadvantages
  - Client: "I know this is what I asked for, but this is not what I really wanted"
  - Heavily documentation dependent – too much overhead for small software
- When to Deploy?
  - Large software
  - Both customer sand developers have the same expectation about the end product – no requirement change

# The Prototyping Model

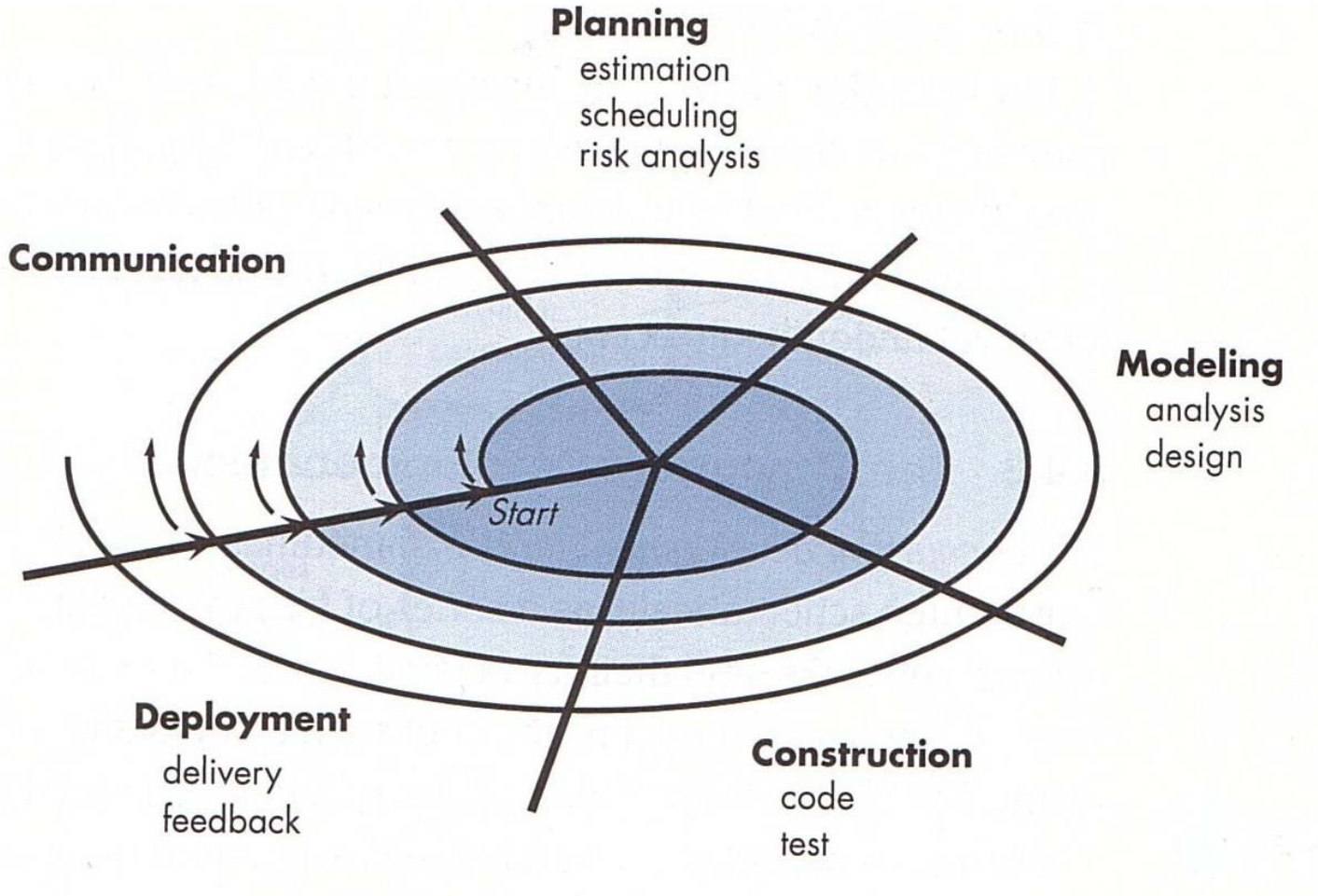



Pressman, 2005

# The Prototyping Model – Advantages and Disadvantages

- Advantages
  - Better communication and avoid returning to its previous phases (Waterfall)
  - An early prototype to have a common understanding (between) developer and customer ) about the requirements identification
- Disadvantages
  - Require to build the rapid prototype as early as possible
  - The prototype building effort is wasted if it is not included in the actual product
- When to Deploy?
    - Answered above!

# Spiral Process Model




**Dimensions:**
- **Radial**
- **Angular**
- **Quadrants**

***Pressman, 2005***

# Spiral – Advantages and Disadvantages

- Advantages
  - Minimizes development risks by using both prototype and risk analysis
  - It is a waterfall model with each phase is preceded by risk analysis
- Disadvantages
  - Useful only for a software system within an organization – difficult to abandon a project of an external client even it is too risky
  - For small software projects risk analysis cost may be too high compared to the total project cost
  - Improper risk analysis may lead to unsuccessful projects
- When to Deploy?
  - Answered above!

# Quality of a Process

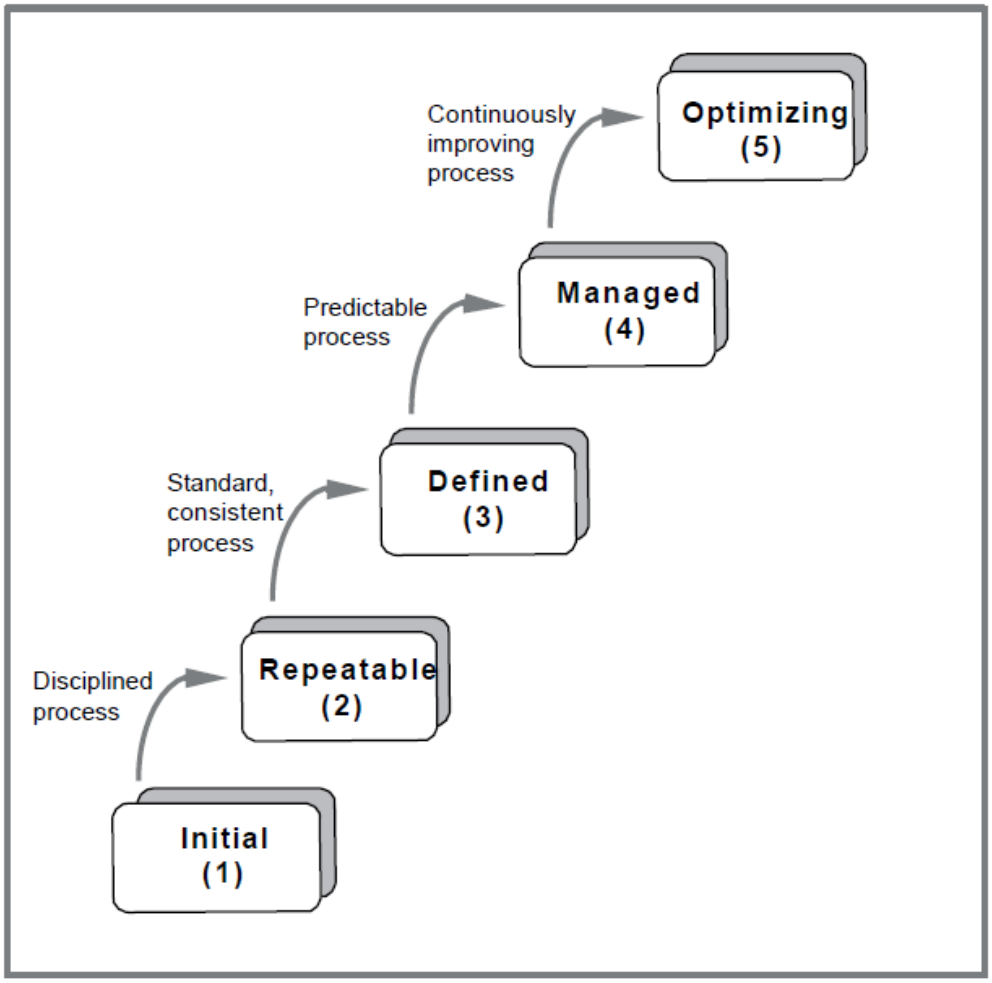
- A process is required to achieve a quality product
- But what about the quality of a process?
  - ISO 9000
  - CMM

# ISO 9000 Standard

- Documentation-based: every step of each process must be documented in a specific form
  - A set of guidelines for quality assurance
  - Better documentation leads to better process/product
- ISO 9000-3
  - Standards for software development, operation, and maintenance
  - Specifies 20 elements (separate requirements for each element such as management , design , and quality assurance issues )

# Capability Maturity Model (CMM)

- CMU/SEI-93-TR-024 ESC-TR-93-177



# Capability Maturity Model – Contd.

- CMM (SEI, CMU), until December 2007
  - Evaluate an organization's software process against an 85-item questionnaire
  - Used to improve the software process independent of the process model used
- CMMI: Capability Maturity Model Integration (SEI, CMU, Jan 2008)
  - Nearly all CMM concepts are incorporated into the CMMI
  - Some existing process areas were modified and some were newly added
  - Implementation goal was added that applies to each process area
  - A continuous (iterative) representation is possible as well as the previous staged representation

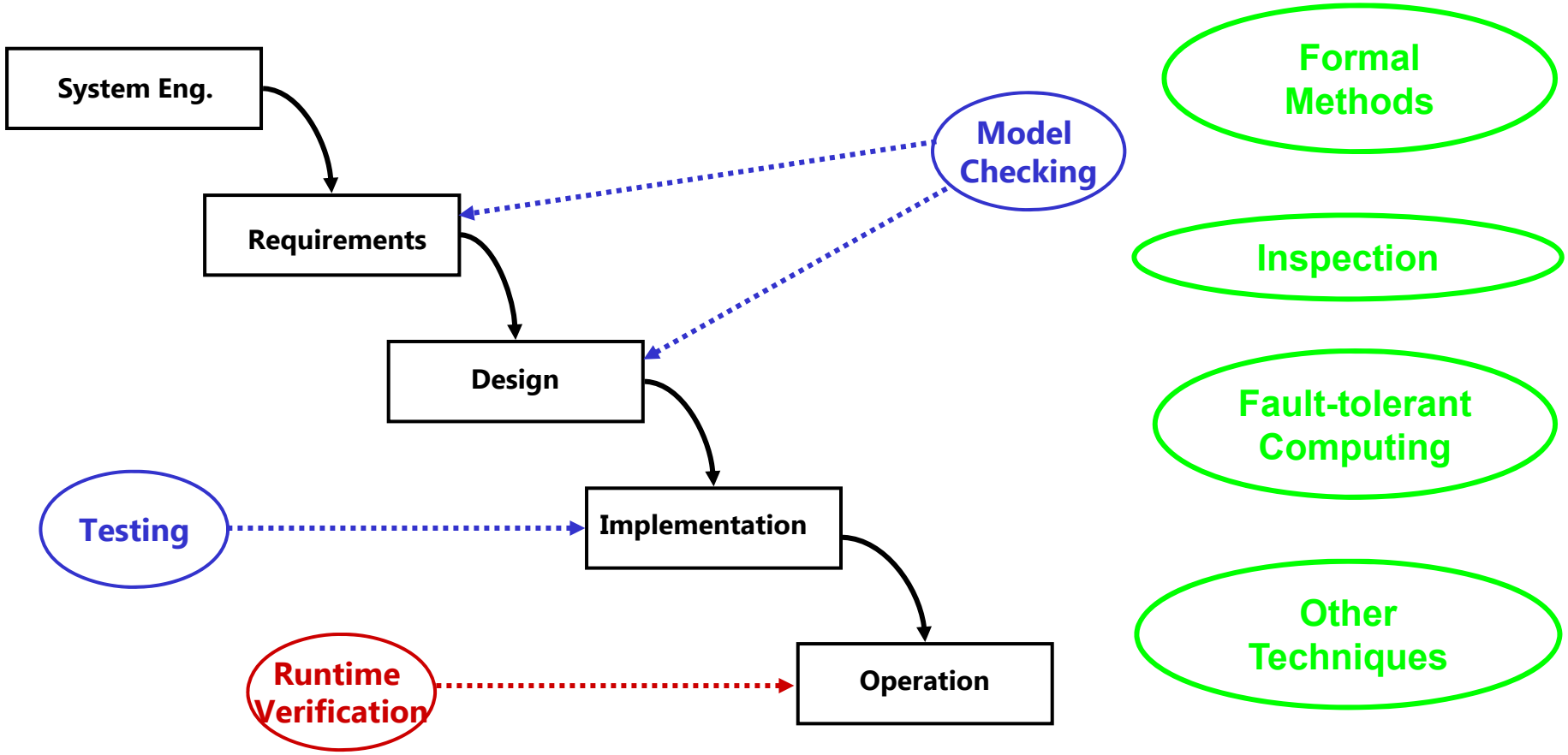
# Capability Maturity Model – Contd.

- Level 1
  - Basically no process – unpredictable cost, schedule, and quality
- Level 2
  - Cost and quality vary highly but somehow controlled
  - Applies some ad hoc processes and methods
- Level 3
  - Qualitative – reliable costs and schedules
  - Applies defined processes for software product management
- Level 4
  - Quantitative – reasonable statistical control of product quality
  - Applies process measurement and analysis
- Level 5
  - Quantitative automation and continuous improvement
  - Applies process change management and current tools and techniques

# Methods for Reliable Software

- Methods for Reliable Software
  - Comparative discussions on different reliability improvement techniques deployed in software life cycles
- Different but complementary techniques
  - Formal methods
  - Testing
  - Inspection
  - Runtime verification
  - Fault-Tolerant computing
  - …

# Reliability Improvement Effort in Software Life Cycle



# Methods for Reliable Software – Formal Methods

- Formal methods include
  - Model checking
  - Theorem proving
  - Many other (formal) mathematical methods for specification and verification
- Primarily safety critical systems (e.g., health care equipment, aerospace)
- Requires in depth mathematical knowledge
- Precise but very expensive and slow process

# Methods for Reliable Software – Formal Methods

- Model Checking
  - Automatically generates and checks all reachable states of the model with respect to a set of properties (requirements specifications)
  - May provide a counter execution to show when the model does not satisfy a given property
- Theorem proving
  - Use formulas and inductions rather than searching a state space
  - More scalable in handling large systems (uses inference)
  - Does not provide counter example and may be manual
- Both usually work on a model (instead of implementation) and check if the model satisfies certain properties

# Methods for Reliable Software – Verification and Validation

- Verification
  - Prove that a product meets its specification
  - Are we building the product right?
  - Example: formal verification, correctness proof
- Validation
  - Experiment to show that its certain requirements are met
  - Are we building the right product?
  - Example: simulation, testing
- Both "verification" and "validation" are used interchangeably or for completely different meanings

# Methods for Reliable Software – Testing

- Execute the software implementation for a set of inputs or scenarios and evaluating the results based on
  - Requirements (customer acceptance testing)
  - Specification and design (functionality and interface testing)
  - Algorithmic logic (control path testing)
  - Execution history (regression testing)
  - …
- Systematic testing steps
  - Create and select test cases
  - Execute the tests and record the results
  - Evaluate the results
  - Select the test completeness criteria

# Methods for Reliable Software – Testing

- Passive Testing
  - Expect not to disturb the normal operation of software
  - Verify whether the I/O meets the specification
- Active Testing
  - Generate test cases from requirements, design, etc.
  - Execute the system to evaluate and compare the outputs with the expected ones
- Testing/Debugging
  - Testing shows that a program has defects (bugs) and it fails
  - Debugging identifies the cause of the failure and tries to remove the cause
- Test Oracle
  - An oracle defines the expected outputs with respect to inputs

# Methods for Reliable Software – Testing

- "Program testing can be used to show the presence of bugs, but never to show their absence" [Dijkstra 1972]
  - a program tested only for positive numbers may not work with negative numbers
- As a result, test coverage is an important issue
- Generation and application of test cases to test everything is difficult (nearly impossible)
- Testing may become less effective if the expected operational profile changes at run time

# Summary

- Software Quality
  - Software quality equation
  - Software quality parameters
- Software Process Models
  - The Waterfall Model
  - The Rapid Prototyping
  - The Spiral Model
  - Software Process Evaluation – Capability Maturity Model,
  - ISO 9000
- Methods for Reliable Software

# Lecture Sources


- Jeffrey Voas, "Trusted Computing's Holy Grail," DSN 2002.
- Object-Oriented & Classical Software Engineering Stephen R. Schach, McGraw-Hill Companies, 1998
- Software Engineering, Sommerville, Addison Wesley, 1996
- Upgrading from SW-CMM to CMMI, Technical Report, Software Engineering Institute, Carnegie Mellon University, Pennsylvania, USA.
- Software Engineering: A Practitioner's Approach, 6/e, Roger S Pressman, 2005.
- Paulk, Mark C.; Weber, Charles V; Curtis, Bill; Chrissis, Mary Beth (February 1993). "Capability Maturity Model for Software, Version 1.1". *Technical Report* (Carnegie Mellon University / Software Engineering Institute). CMU/SEI-93-TR-024 ESC-TR-93-177.
- Software QA and Testing Resource Center, 2002 http://www.softwareqatest.com/
- Object-Oriented & Classical Software Engineering Stephen R. Schach, McGraw-Hill Companies, 1998
- Software Engineering, Sommerville, Addison Wesley, 1996
- Formal Methods Publications, 1995: http://www.comlab.ox.ac.uk/archive/formal-methods/pubs.html
- J. Cordy, Software Quality Assurance Course Notes, 2002