Code Inspection Checklist

- Variables
  - Meaningful names?
  - Named constants instead of hard-coded numbers
  - Are read-only variables declared const?
  - Are all variables used?
- Functions
  - Meaningful names
  - Are all parameters used?

Code Inspection Checklist (cont.)

- Correctness
  - Parentheses and brackets properly matched?
  - Switch cases all end in break? Have default?
- Initialization
  - For all variables before use
- Loops
  - All terminate?
  - Do break and continue statements work correctly?
  - Are loop control variables modified in loop body?

Code Inspection Checklist (cont.)

- Dynamic allocation
  - All allocations properly deallocated?
- Pointers
  - Can a NULL pointer be dereferenced?
- Comments
  - Are comments appropriate?
  - Do comments accurately describe the code?
- Defensive programming
  - Are checks made for divide by zero, invalid data, etc.?

Code Walkthroughs or Reviews

- Less formal than inspections
- Typically fewer people involved
- Discussions of alternatives are allowed
  - Even encouraged

Software Engineering

COEN 174

Ron Danielson

Testing and Quality Assurance

Chapter 10

Objectives

- Understand why quality is essential in software engineering
- Know basic techniques for testing software, including test coverage
- Understand basic techniques for software validation and verification

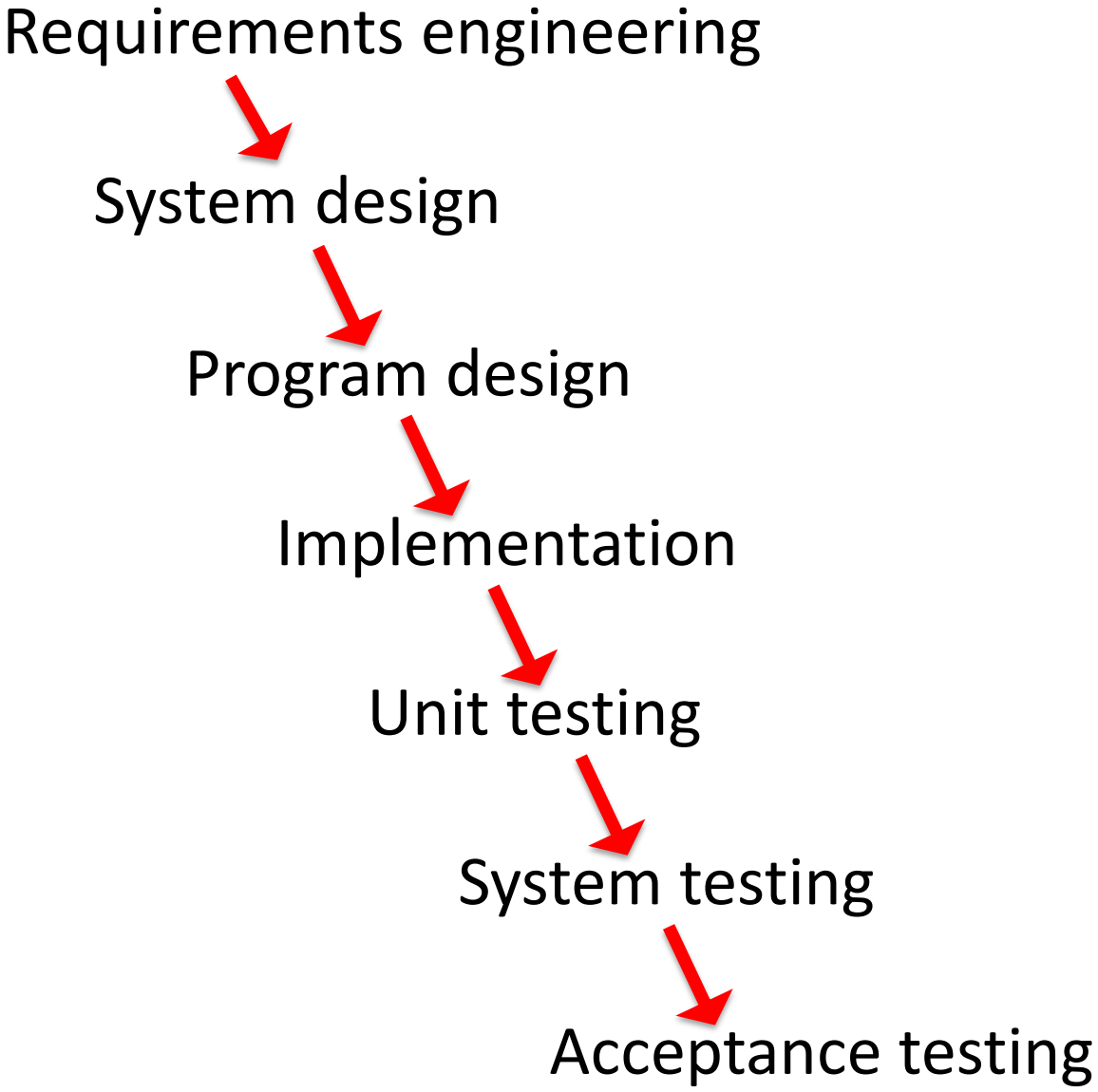
Ensuring Quality

- **Quality Assurance**: actions to **measure** and **improve** quality in a ***product*** and a ***process***
  - Most large SW development organizations have separate QA organizations that do more elaborate levels of testing
- **Quality Control**: actions to **validate** and **verify** the quality of a product
  - By detecting faults and fixing the defects

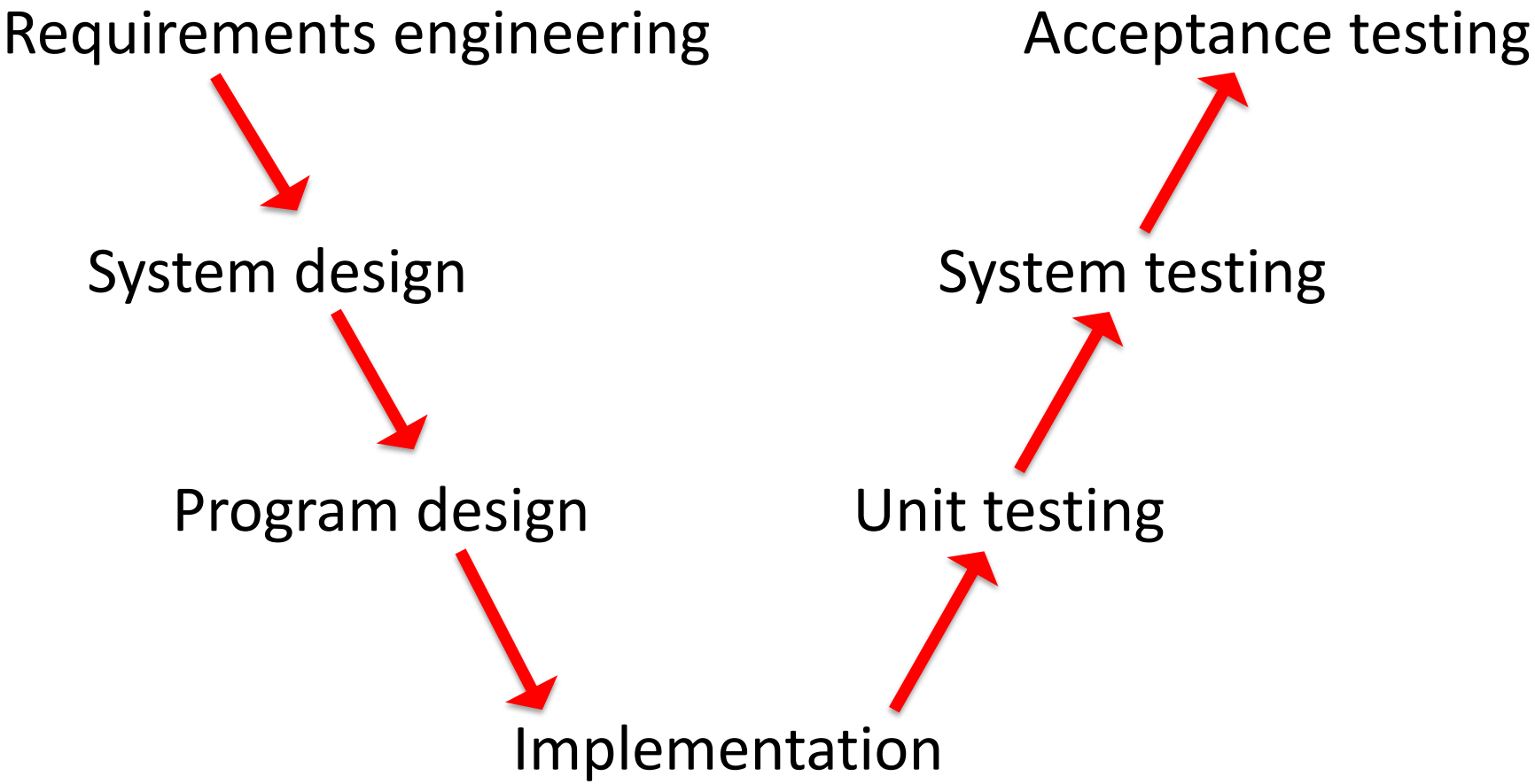
What Is Software Quality

- Traditionally two measures
  - Conforms to requirements
  - Serves the purpose for which the SW is intended
- **Verification** is checking conformance
  - Did SW evolve properly from requirements
    - i.e., does the SW work correctly?
  - Verified against design and written specification
- **Validation** is checking that software meets user requirements
  - Was the correct system built?
  - Validated with client

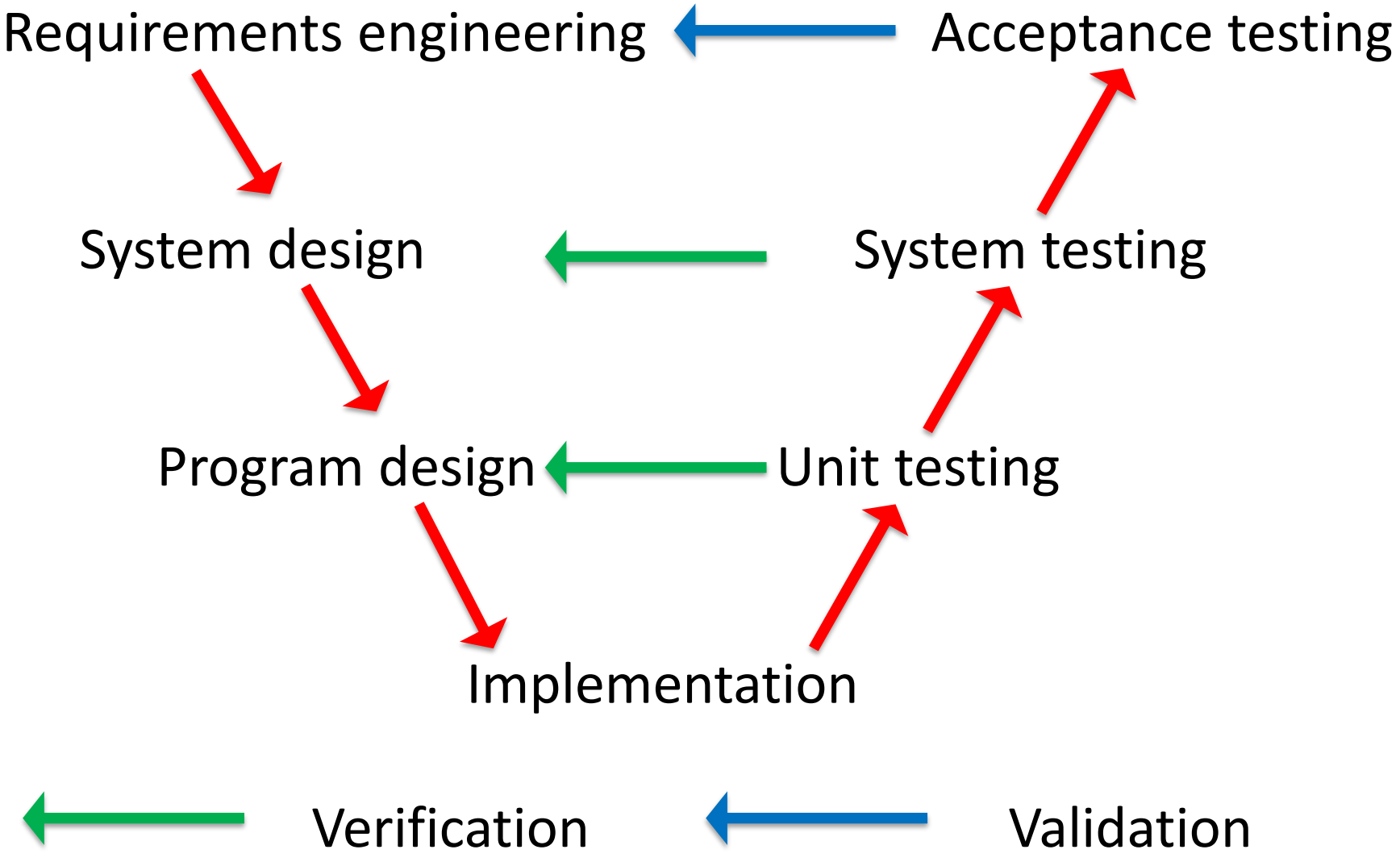
SE Process


Requirements engineering
System design
Program design
Implementation
Unit testing
System testing
Acceptance testing

SE Process


Requirements engineering
System design
Program design
Implementation
Unit testing
System testing
Acceptance testing

SE Process


Requirements engineering
Acceptance testing
System design
System testing
Program design
Unit testing
Implementation
Verification
Validation

Verification vs. Validation

- Verification: Did we **build the product right**?
- Validation: Did we **build the right product**?

Lack of Quality

- Caused by errors
- **Fault** or **defect** is a condition that may cause a failure
  - Caused by errors made by SEs
  - Error may be in code
  - Error may be in requirements, or in design documents
- A **failure** is the inability of a system to perform as required by the specification
- **Severity** of a fault or failure is determined by the consequences
- **Priority** of a fault or failure is determined by importance of finding a fix

Finding Errors

- **Testing** is running code *in a controlled environment* to check that it behaves correctly
  - Often easiest, usually most common, only dynamic way to find errors
- Formally prove software is correct
- Static analysis
  - Informal reviews
  - Walkthroughs
  - Technical reviews
  - Inspections

increasing formality

Testing

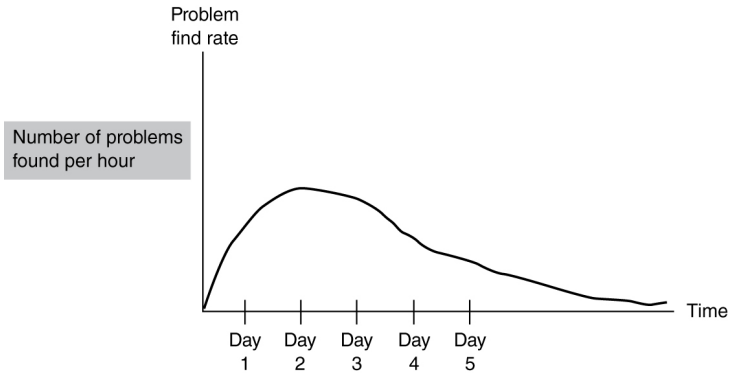
- Testing is an activity performed to
  - Find faults
    - Can't prove there are no faults since exhaustive testing is impossible
      - Test finding roots of quadratic equation $ax^2 + bx + c$
      - For 32-bit integer values, 1 test/ns, 2.5 trillion years
  - Assess product quality
    - Fewer errors found implies higher quality
    - Quality of test cases significant factor in accuracy of this assessment
- Test early, test often

Testing (cont.)

- Levels of testing
  - **Unit** (single methods, procedures, modules, …)
  - **Functional** (multiple units as a functional unit)
  - **Component** (multiple functions)
  - **Integration/system** (all components together)
- Who does testing
  - Programmers (unit)
  - Testers/QA staff (integration/system)
  - Users (alpha and beta)
  - Clients (acceptance)

Testing (cont.)

- When to stop testing
  - When all test cases have been passed
  - When all errors have been found
  - When error detection levels out



  - Based on fault seeding
    - Embed known errors in software product
    - At any point, percentage of seeded errors found is an estimate for percent of real errors found

Testing (cont.)

- Other criteria for stopping testing
  - Tester can't find error in N minutes
  - **Nominal**, **boundary**, and **out-of-bounds** tests run error-free
  - When agreed checklist of test types has been completed
  - When time scheduled for testing expires