Introduction to Software Engineering

Kenneth M. Anderson
University of Colorado, Boulder
CSCI 5828 — Lecture 2 — 01/17/2008

Credit where Credit is Due

- Some text and images for this lecture come from the lecture materials provided by the publisher of the Pfleeger/Atlee textbook. As such, some material is copyright © 2006 Pearson/Prentice Hall.

Lecture Goals

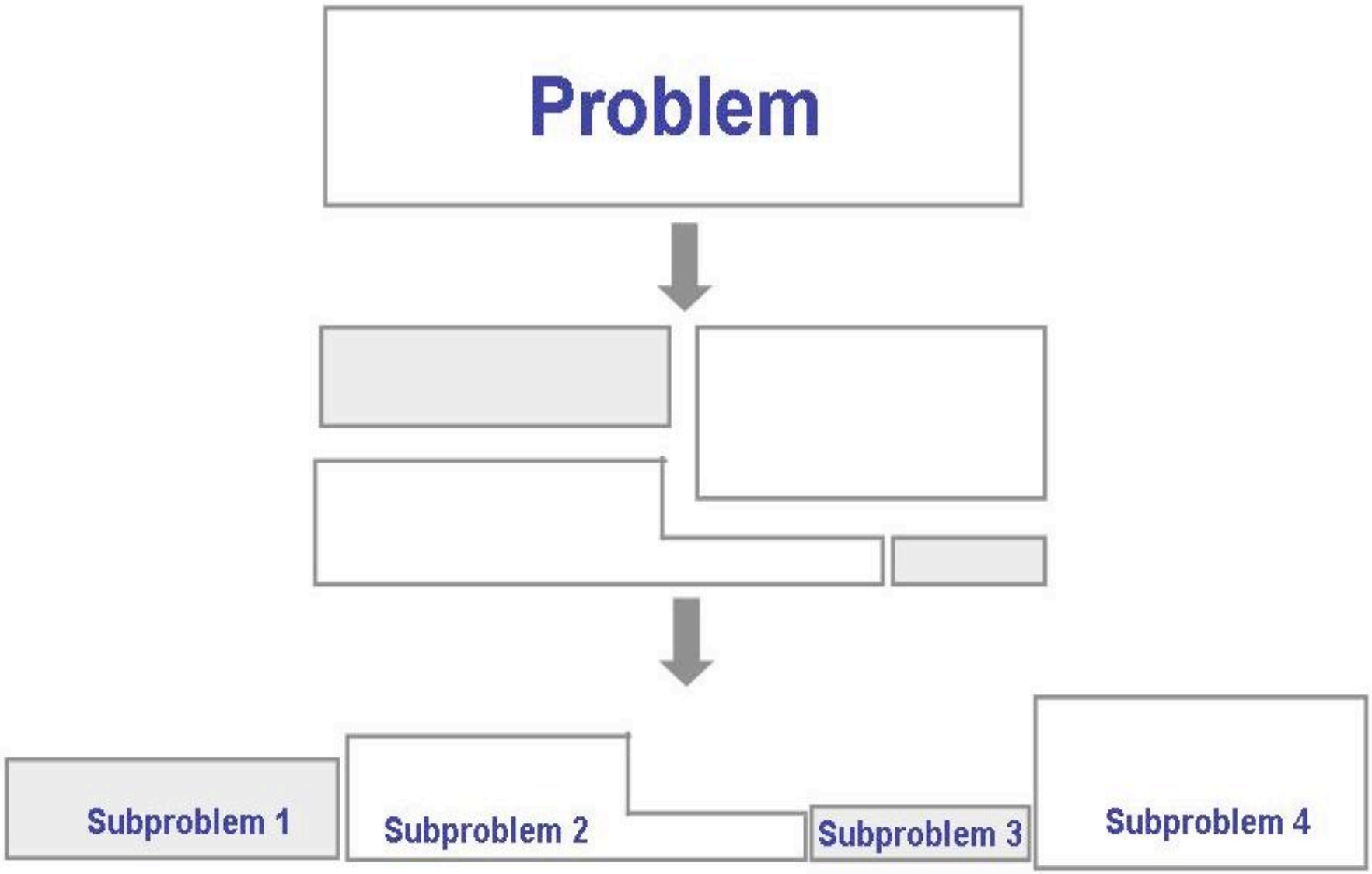
- Review material in Chapter 1 of the Pfleeger/Atlee book
  - What do we mean by software engineering?
  - Examine SE's track record
  - What do we mean by good software?
  - Examine the "system's approach" to building software
  - Examine how SE has changed over the past three decades

What is Software Engineering?

- Simply Put: Its about solving problems with software-based systems
  - Design and development of these systems require
    - Analysis
      - decomposing large problems into smaller, understandable pieces
        - abstraction is the key
    - Synthesis
      - building large software systems from smaller building blocks
        - composition is challenging

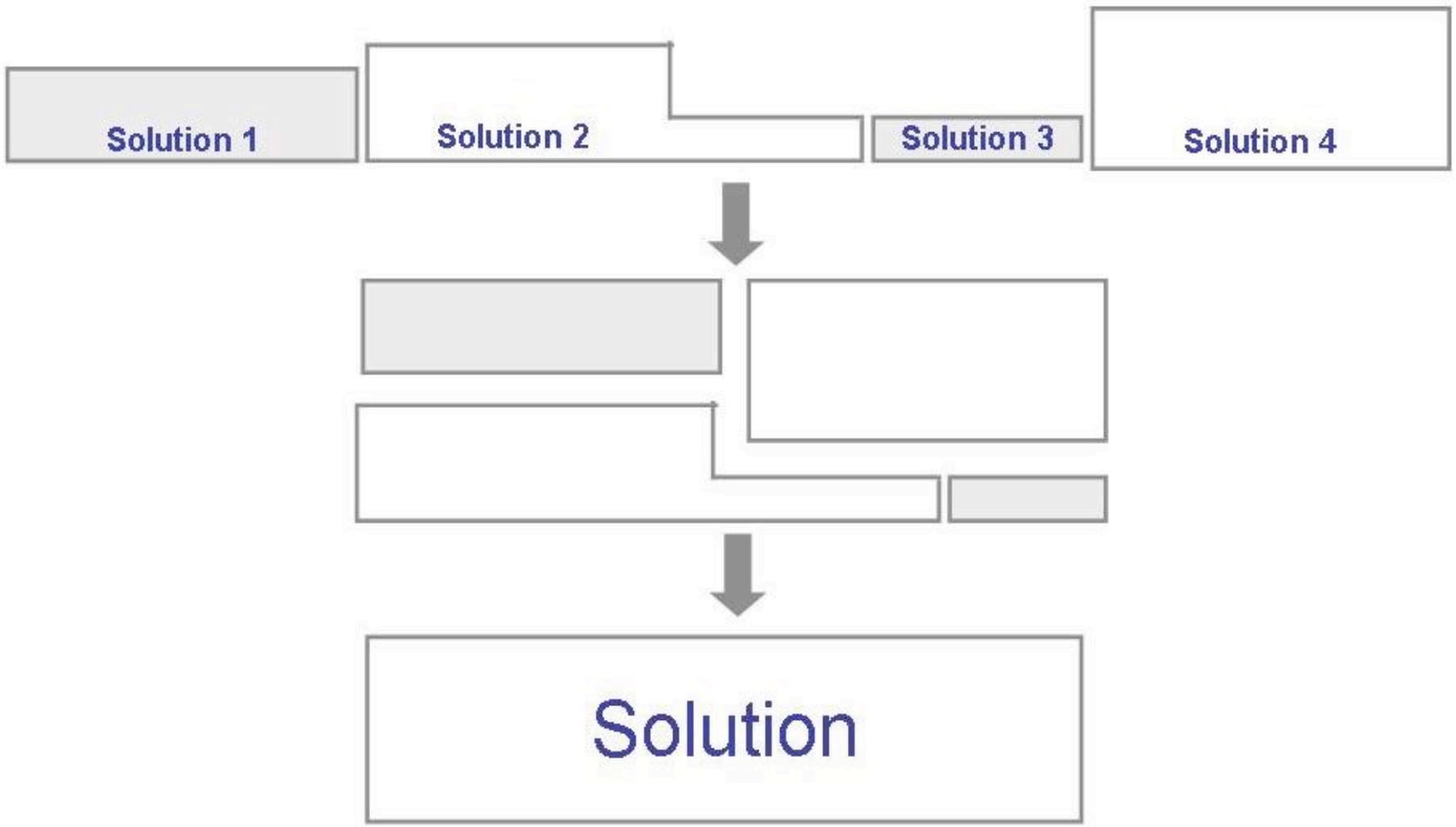
Solving Problems (I)

- The analysis process



Solving Problems (II)

- The synthesis process



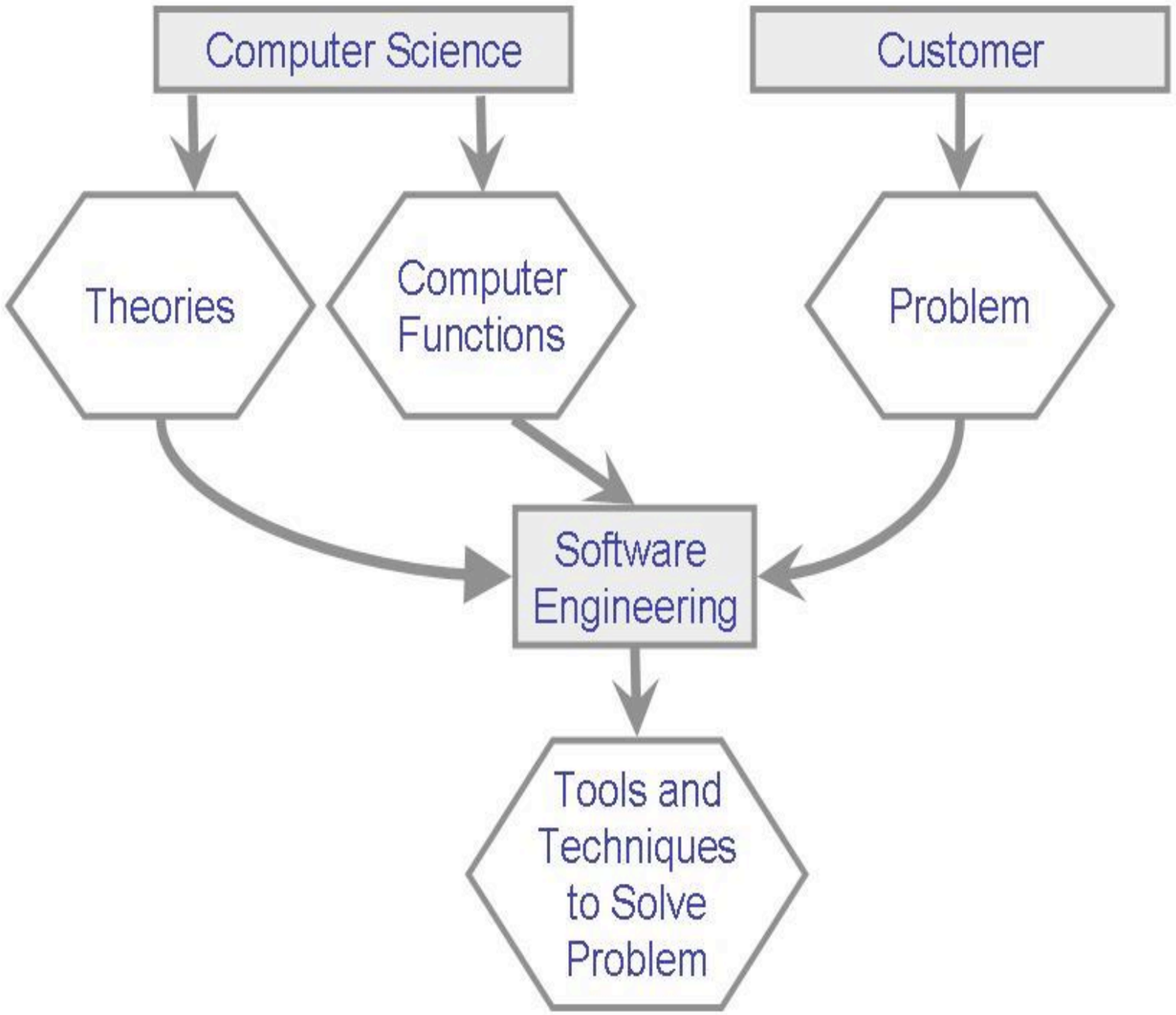
Solving Problems (III)

- To aid us in solving problems, we apply
  - **techniques**: a formal “recipe” for accomplishing a goal that is typically independent of the tools used
    - Example: procedure for thickening a sauce without causing it to curdle
  - **tools**: an instrument or automated system for accomplishing something in a better way, where “better” can mean more efficient, more accurate, faster, etc.
  - **procedures**: a combination of tools and techniques that, in concert, produce a particular product
  - **paradigms**: a particular philosophy or approach for building a product
    - Think: “cooking style”: may share procedures, tools, and techniques with other styles but apply them in different ways
    - Example: OO approach to development vs. the structured approach

Relationship to Computer Science (I)

- How does Software Engineering relate to the discipline of Computer Science
  - Computer Science is a **scientific discipline** that focuses on developing new principles, new techniques, new languages, new hardware, etc.
  - Software engineering is an engineering discipline that focuses on using software and computing technology as problem solving tools
    - It draws upon the techniques that computer science develops (along with other disciplines) to aid in the process of solving those problems

Relationship to Computer Science (II)



Software Engineering: The Good

- Software engineering has helped to produce systems that improve our lives in numerous ways
  - helping us to perform tasks more quickly and effectively
  - supporting advances in medicine, agriculture, transportation, and other industries
- Indeed, software-based systems are now ubiquitous
  - How many computers do you have in your home?
  - How many times do you interact with a software-based system each day?

Software Engineering: The Bad (I)

- Software is not without its problems
  - Systems function, but not in the way we expect
  - Or systems crash, make mistakes, etc.
  - Or the process for producing a system is riddled with problems leading to a failure to produce the entire system
    - many projects get cancelled without ever producing a system
- One study in the late 80s found that in a survey of 600 firms, more than 35% reported having a runaway development project. A runway project is one in which the budget and schedule are completely out of control.

Software Engineering: The Bad (II)

- CHAOS Report from Standish Group
  - Has studied over 40,000 industry software development projects over the course of 1994 to 2004.
  - Success rates (projects completed on-time, within budget) in 2004 was 34%, up from 16.2% in 1994
  - Failure rates (projects cancelled before completion) in 2004 was 15%, down from 31% in 1994.
  - In 2004, “challenged” projects made up 51% of the projects included in the survey.
    - A challenged project is one that was over time, over budget and/or missing critical functionality

Software Engineering: The Bad (III)

- Most challenged projects in 2004 had a cost overrun of under 20% of the budget, compared to 60% in 1994
- The average cost overrun in 2004 was 43% versus an average cost overrun of 180% in 1994.
- In 2004, total U.S. project waste was 55 billion dollars with 17 billion of that in cost overruns; Total project spending in 2004 was 255 billion
- In 1994, total U.S. project waste was 140 billion (80 billion from failed projects) out of a total of 250 billion in project spending
- So, things are getting better (attributed to better project management skills industry wide), but we've still got a long way to go!
    - 66% of the surveyed projects in 2004 did not succeed!

Software Engineering: The Ugly (I)

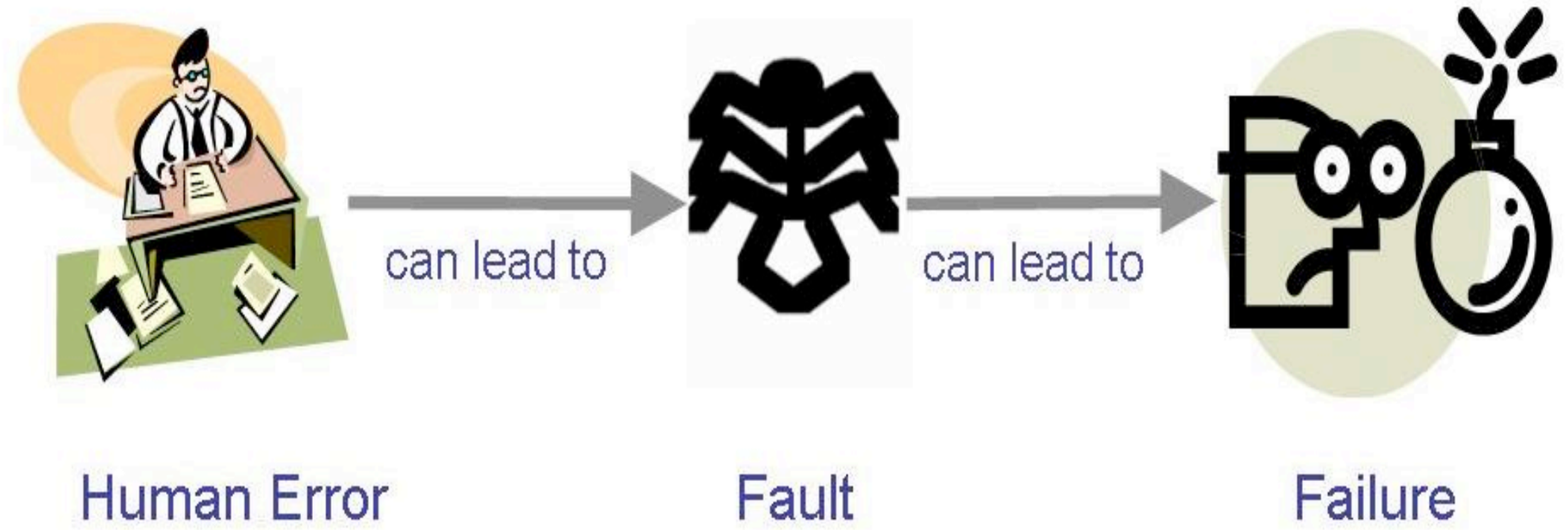
- Loss of NASA's Mars Climate Observer
  - due to conversion error of English and Metric units!
  - even worse: problem was known but politics between JPL and Houston prevented fix from being deployed
- Leap-year bug
  - A supermarket was fined $1000 for having meat around 1 day too long on Feb. 29, 1988
- Denver International Airport
  - Luggage system: 16 months late, 3.2 billion dollars over budget!

Software Engineering: The Ugly (II)

- IRS hired Sperry Corporation to build an automated federal income tax form processing process
  - An extra $90 M was needed to enhance the original $103 product
  - IRS lost $40.2 M on interests and $22.3 M in overtime wages because refunds were not returned on time
- Therac-25 (safety critical system: failure poses threat to life or health)
  - Machine had two modes: “electron beam” and “megavolt x-ray”
  - “megavolt” mode delivered x-rays to a patient by colliding high energy electrons into a “target”
    - Patients died when a “race condition” in the software allowed the megavolt mode to engage when the target was not in position
    - Related to a race between a “type ahead” feature in the user interface and the process for rotating the target into position

Terminology for Describing Bugs

- An error is a mistake made by a human
- A fault is the manifestation of the error in a software artifact
- A failure is a departure from a system's required (or expected) behavior



What is Good Software? (I)

- "Good" is often associated with some definition of quality. The higher the quality, the better the software.
- The problem? Many different definitions of quality!
  - **Transcendental**: where quality is something we can recognize but not define ("I know it when I see it")
  - **User**: where quality is determined by evaluating the fitness of a system for a particular purpose or task (or set of tasks)
  - **Manufacturing**: quality is conformance to a specification
  - **Product**: quality is determined by internal characteristics (e.g. number of bugs, complexity of modules, etc.)
  - **Value**: quality depends on the amount customers are willing to pay
    - customers adopt "user view"; developers adopt "manufacturing view", researchers adopt "product view"; "value view" can help to tie these together

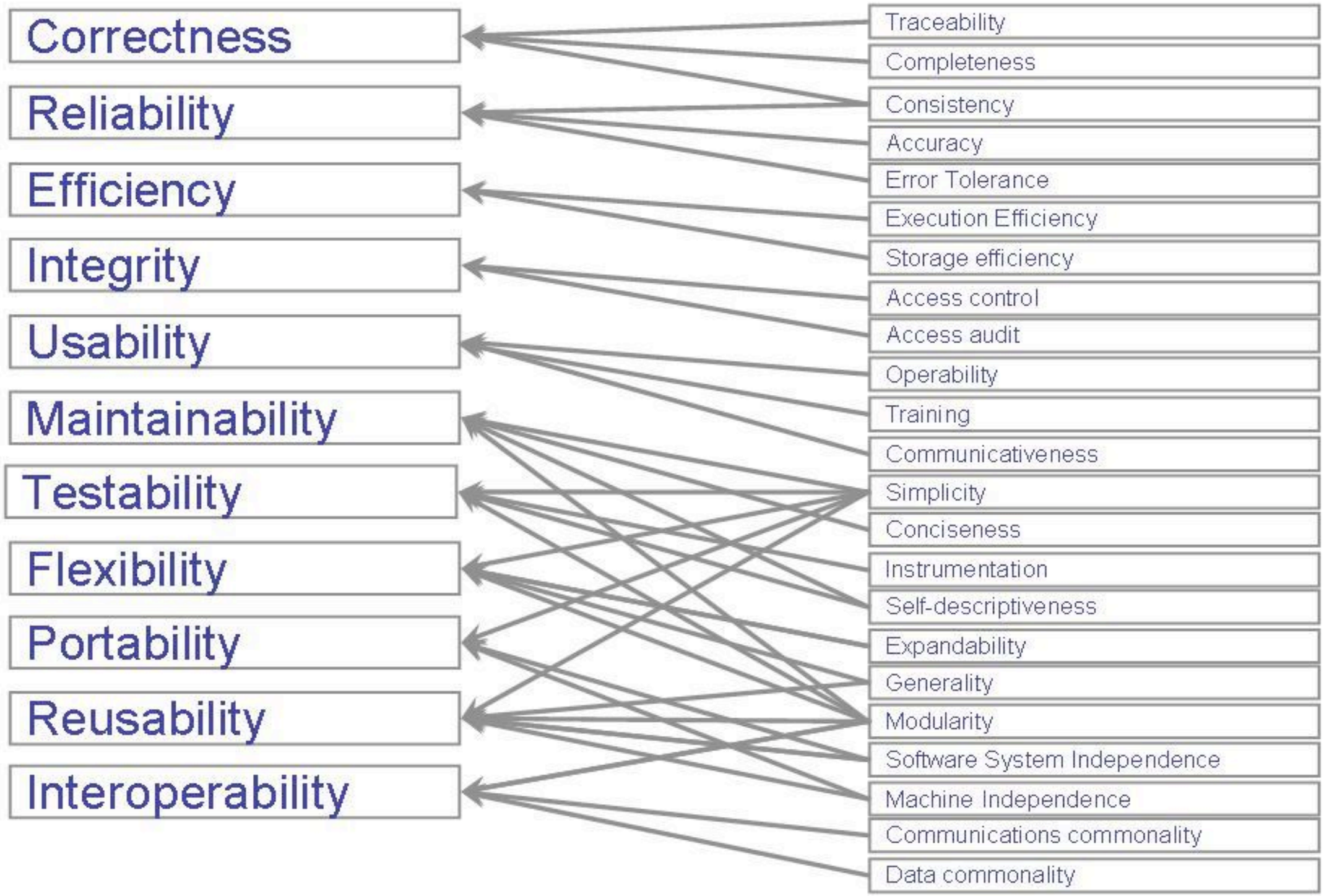
What is Good Software? (II)

- Good software engineering must always include a strategy for producing high quality software
- Three common ways that SE considers quality:
  - The quality of the product (product view)
  - The quality of the process (manufacturing view)
  - The quality of the product in the context of a business environment (user view)
- The results of the first two are termed the “technical value of a system”
  - The latter is termed the “business value of a system”

The Quality of the Product

- Users judge a system on external characteristics
  - correct functionality, number of failures, types of failures
- Developers judge the system primarily on internal characteristics
  - types of faults, reliability, efficiency, etc.
- Quality models can be used to relate the user's external view to the developer's internal view
  - An example is McCall's quality model that relates external software characteristics to internal characteristics
    - This model can be useful to developers: want to increase "reliability" examine your system's "consistency, accuracy, and error tolerance"

McCall's Quality Model



The Quality of the Process

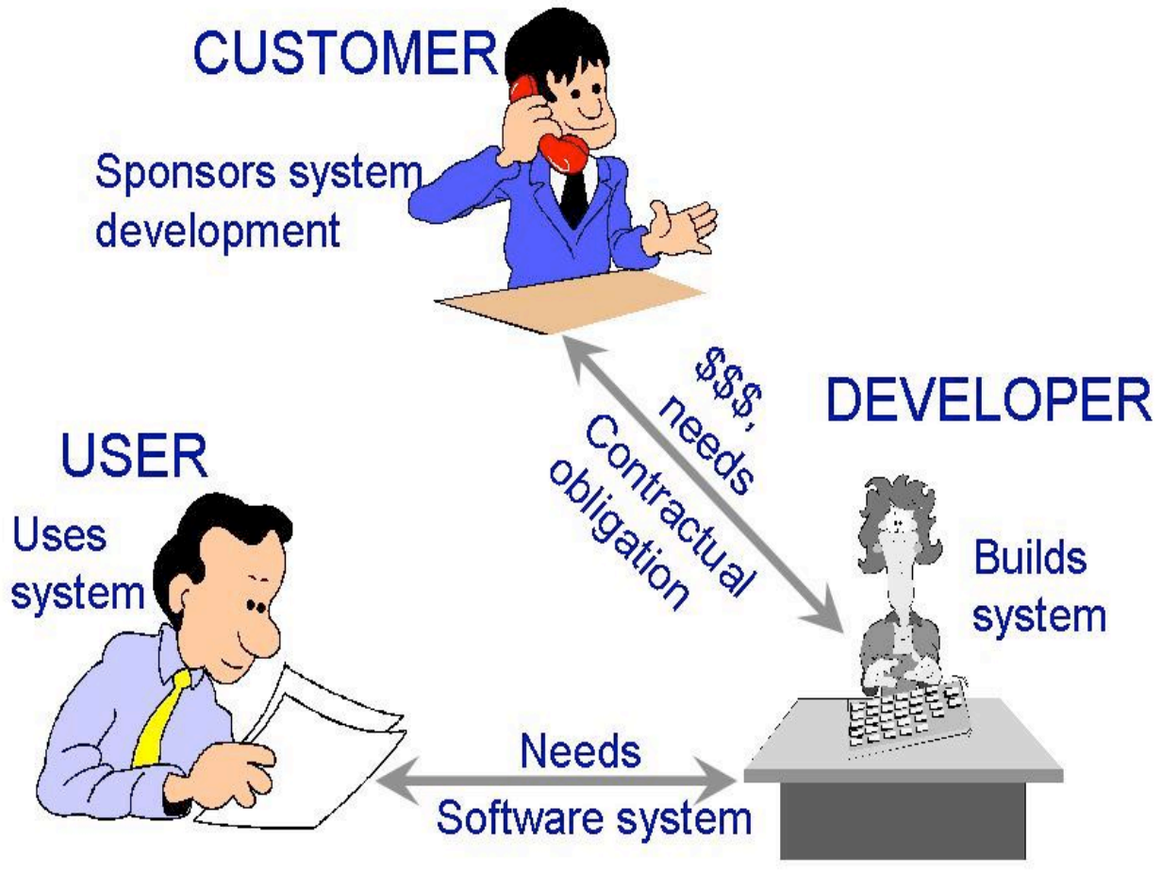
- Quality of the development and maintenance process is as important as the product quality
  - The development process needs to be modeled
- Modeling will address questions such as
  - What steps are needed and in what order?
  - Where in the process is effective for finding a particular kind of fault?
  - How can you shape the process to find faults earlier?
  - How can you shape the process to build fault tolerance into a system?
- Models for Process Improvement (will look at these later)
  - SEI's Capability Maturity Model (CMM)
  - ISO 9000
  - Software Process Improvement and Capability dEtermination (SPICE)

Business Environment Quality

- The business value being generated by the software system
  - Is it helping the business do things faster or with less people?
    - Does it increase productivity?
- To be useful, business value must be quantified
- A common approach is to use the metric “return on investment” (ROI)
- Problem: Different stakeholders define ROI in different ways!
  - Business schools: “what is given up for other purposes”
  - U.S. Government: “in terms of dollars, reducing costs, predicting savings”
  - U.S. Industry: “in terms of effort rather than cost or dollars; saving time, using fewer people”
- Differences in definition means that one organization’s ROI can NOT be compared with another organization’s ROI without careful analysis

Software Engineering Roles

- Customer: the company, organization, or person who pays for the software system
- Developer: the company, organization, or person who is building the software system
- User: the person or people who will actually use the system



A Systems Approach to Software Engineering (I)

- High-Level Overview
  - Identify activities and objects
  - Define the system boundary (critical)
  - Consider nested systems and relationships to other systems

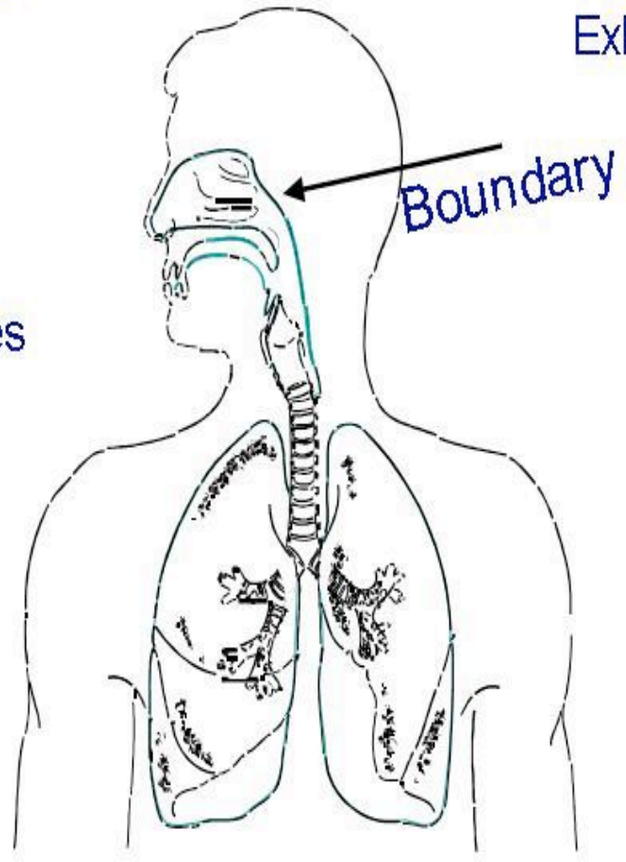
A Systems Approach to Software Engineering (II)

- Activities and objects
  - An activity is an event initiated by a trigger
  - Objects or entities are the elements involved in the activities
- Relationships and the system boundaries
  - A relationship defines the interaction among entities and activities
  - System boundaries determine the origin of input and destinations of the output

A Systems Approach to Software Engineering (III)

- Example System: Human Respiratory System

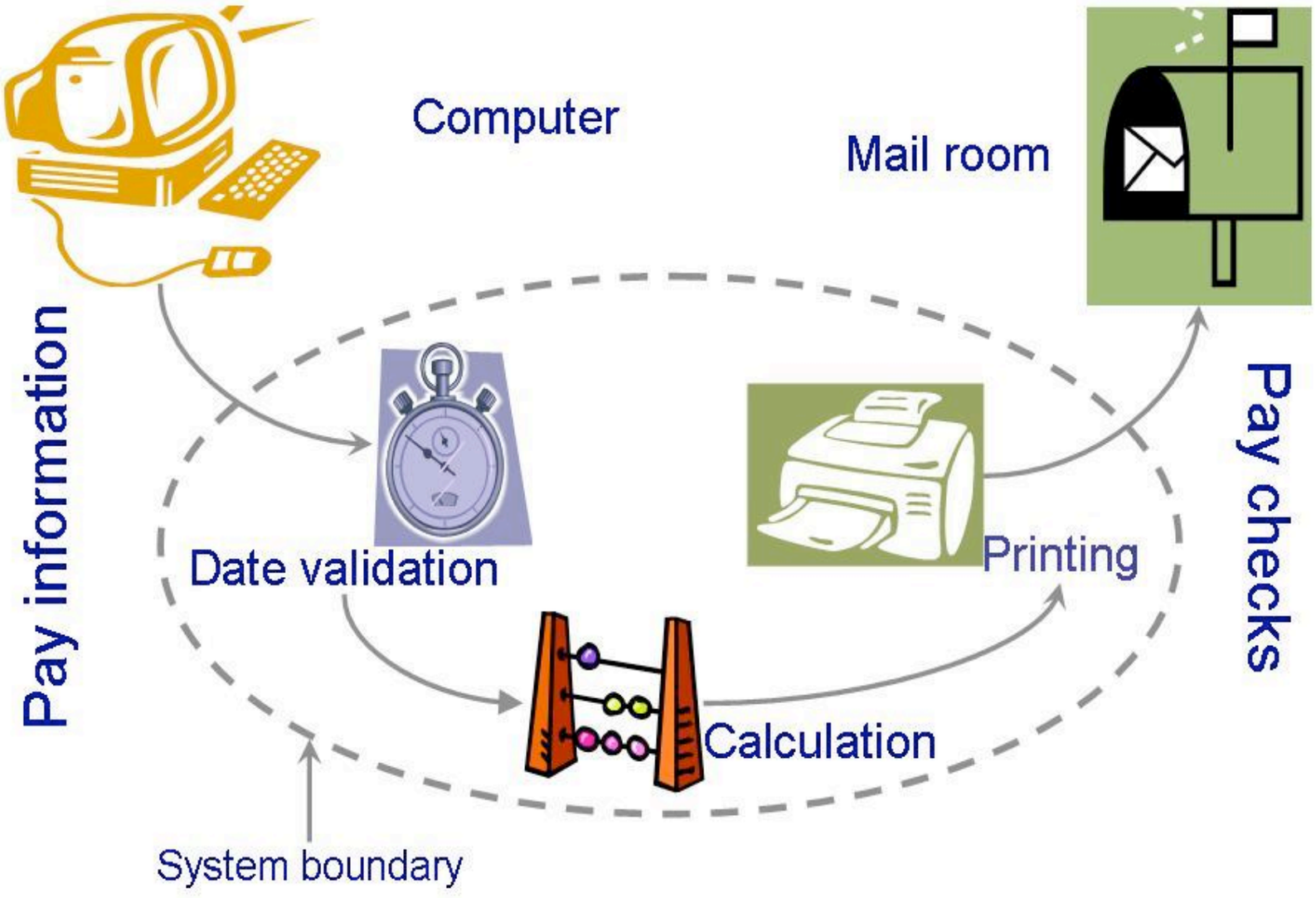
ENTITIES:
Particulate matter
Oxygen
Carbon dioxide
Water
Nitrogen
Nose
Mouth
Trachea
Bronchial tubes
Lungs
Alveoli



ACTIVITIES:
Inhale gases
Filter gases
Transfer molecules to/from blood
Exhale gases

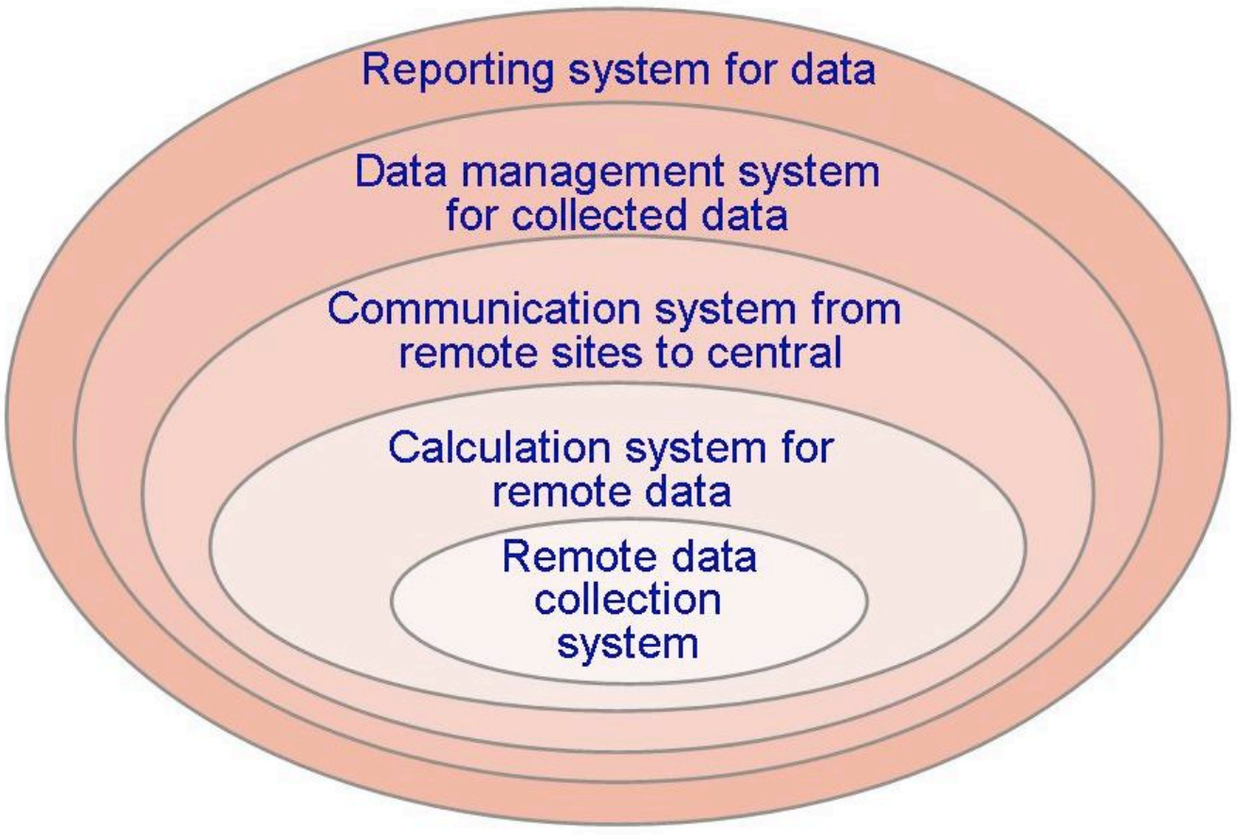
A Systems Approach to Software Engineering (IV)

- Example System: Paycheck Production



A Systems Approach to Software Engineering (V)

- Some systems are dependent on other systems
  - The interdependencies may be complex
- It is possible for one system to exist inside another system
- However, if the boundary definitions are sufficiently detailed, building a larger system from smaller ones is relatively easy



Example of a Layered System

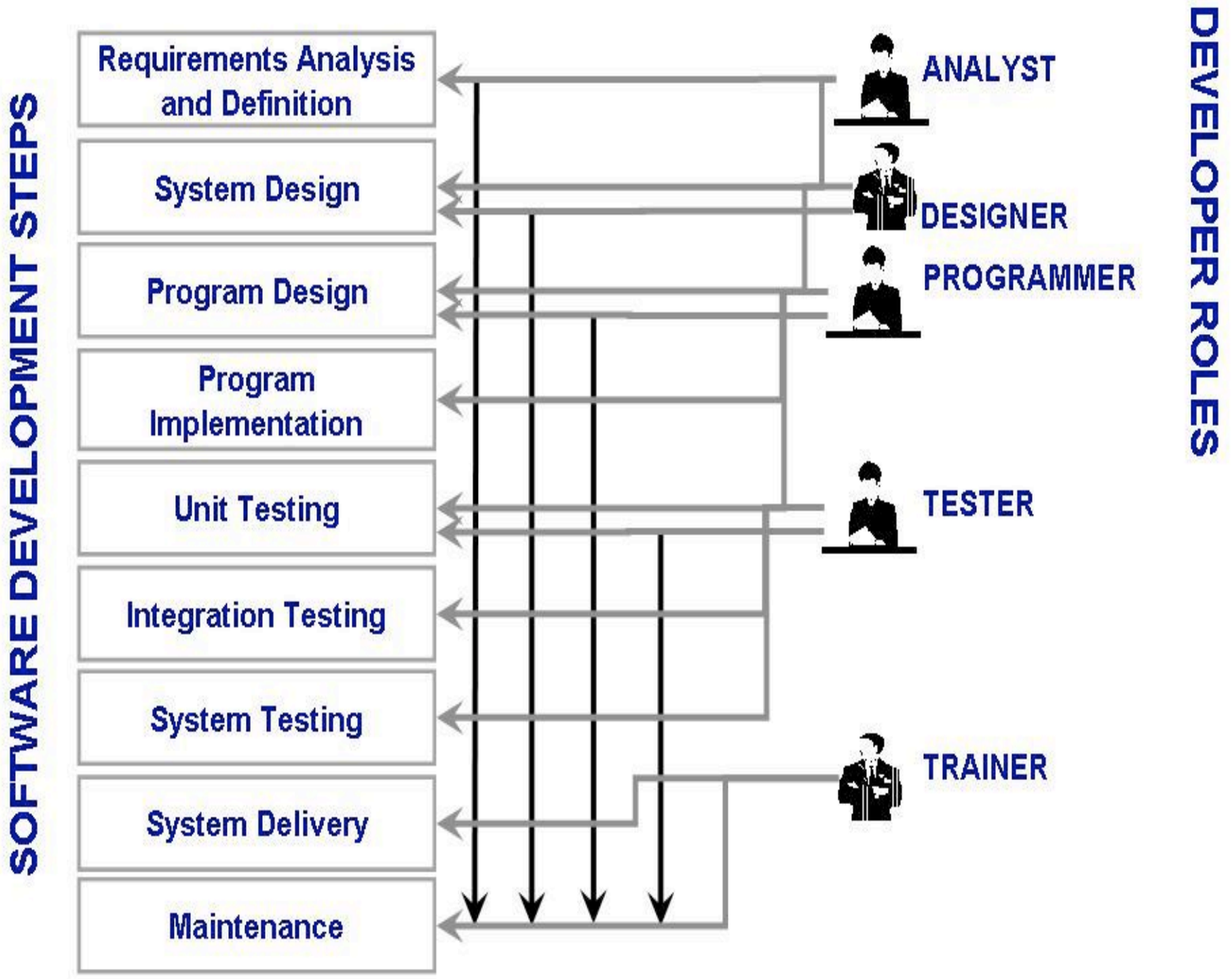
Typical Phases in a Software Development Process

- Requirements analysis and definition
- System design
- Program design
- Writing the programs
- Unit testing
- Integration testing
- System testing
- System delivery
- Maintenance

Typical Members (Roles) of a Development Team

- **Requirement analysts**: work with the customers to identify and document the requirements
- **Designers**: generate a system-level description of what the system us supposed to do
- **Programmers**: write lines of code to implement the design
- **Testers**: catch faults
- **Trainers**: show users how to use the system
- **Maintenance team**: fix faults that show up later
- **Librarians**: prepare and store documents such as software requirements
- **Configuration management team**: maintain correspondence among various artifacts

Mapping between Roles and Phases

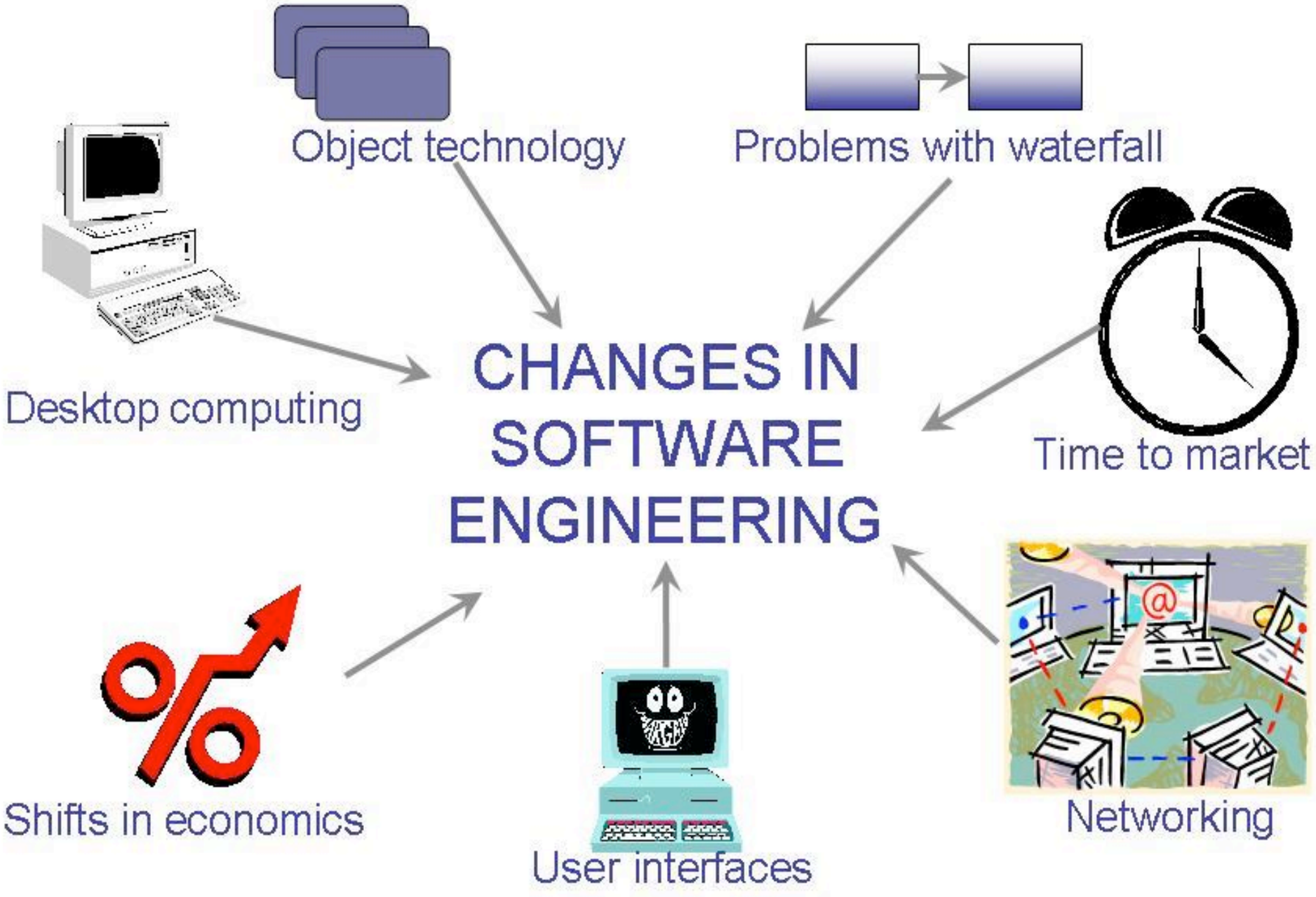


How has Software Engineering Changed? (I)

- Before 1970s
  - Single processors: mainframes
  - Designed in one of two ways
    - as a transformation: input was converted to output
    - as a transaction: input determined which function should be performed
- After 1970s
  - Run on multiple systems
  - Perform multi-functions

How has Software Engineering Changed? (II)

- Wasserman's Seven Key Factors of Change

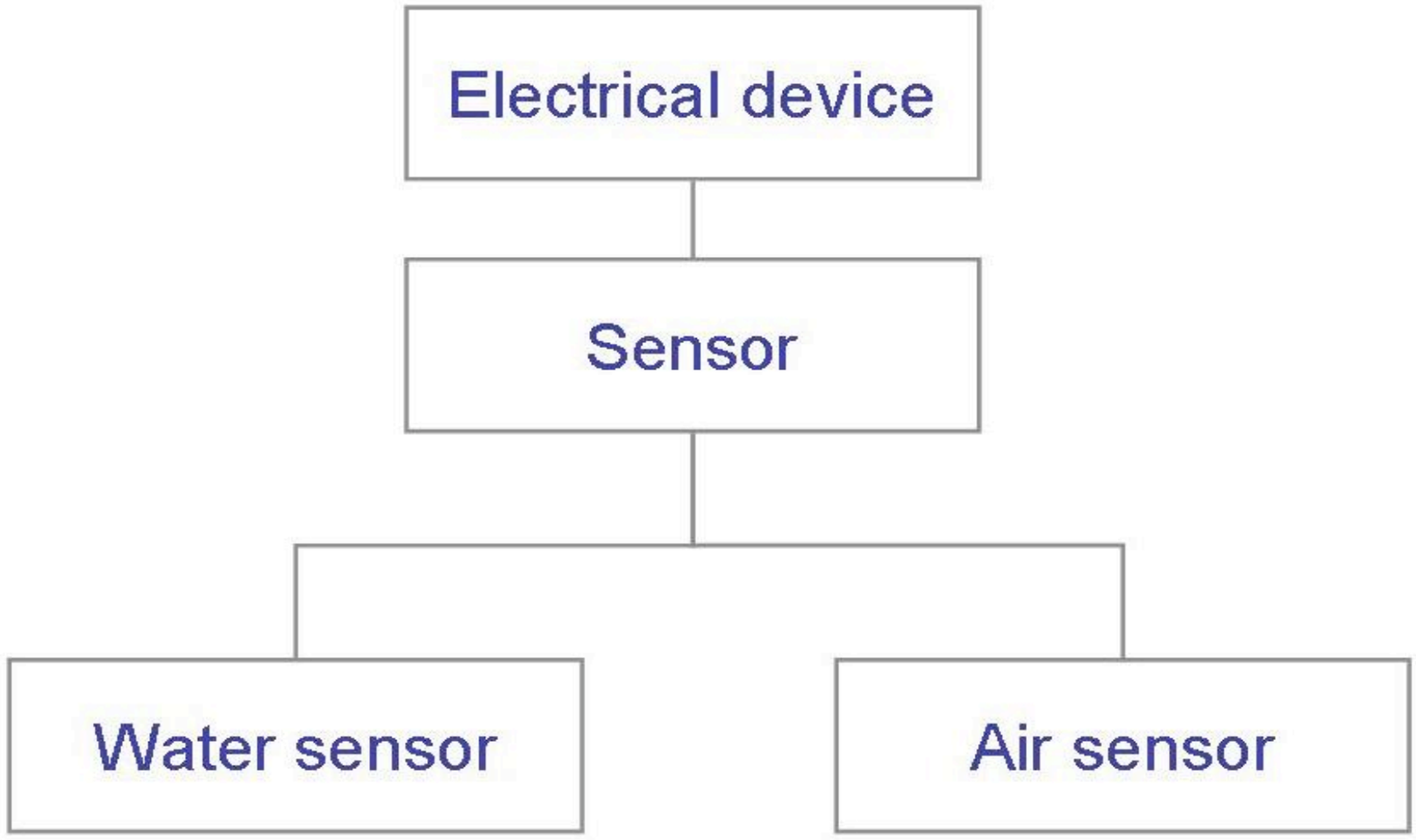


Wasserman's Discipline of Software Engineering

- Abstraction
- Analysis and design methods and notations
- User interface prototyping
- Software architecture
- Software process
- Reuse
- Measurement
- Tools and integrated environments

Abstraction

- A description of a problem at some level of generalization
  - Hide (hopefully lots of) details



A&D Methods and Notations

- Formalize the process of performing analysis and design
- Provide notations for documenting the outcomes of these proceses
- Facilitate communication among developers, customers, and users
- Allow us to build models and check them for completeness and correctness
- Provide materials that can be re-used between projects

Prototyping

- Prototyping: building a small (feature incomplete) version of a system
  - Help users identify key requirements of a system
  - Demonstrate feasibility
- Develop usable user interface via feedback from users

Software Architecture

- A system's architecture describes the system in terms of a set of architectural units and relationships between these units
- Architectural decomposition techniques
    - Modular decomposition
    - Data-oriented decomposition
    - Event-driven decomposition
    - Outside-in-design decomposition
        - based on user inputs to system
    - Object-oriented decomposition

Software Process

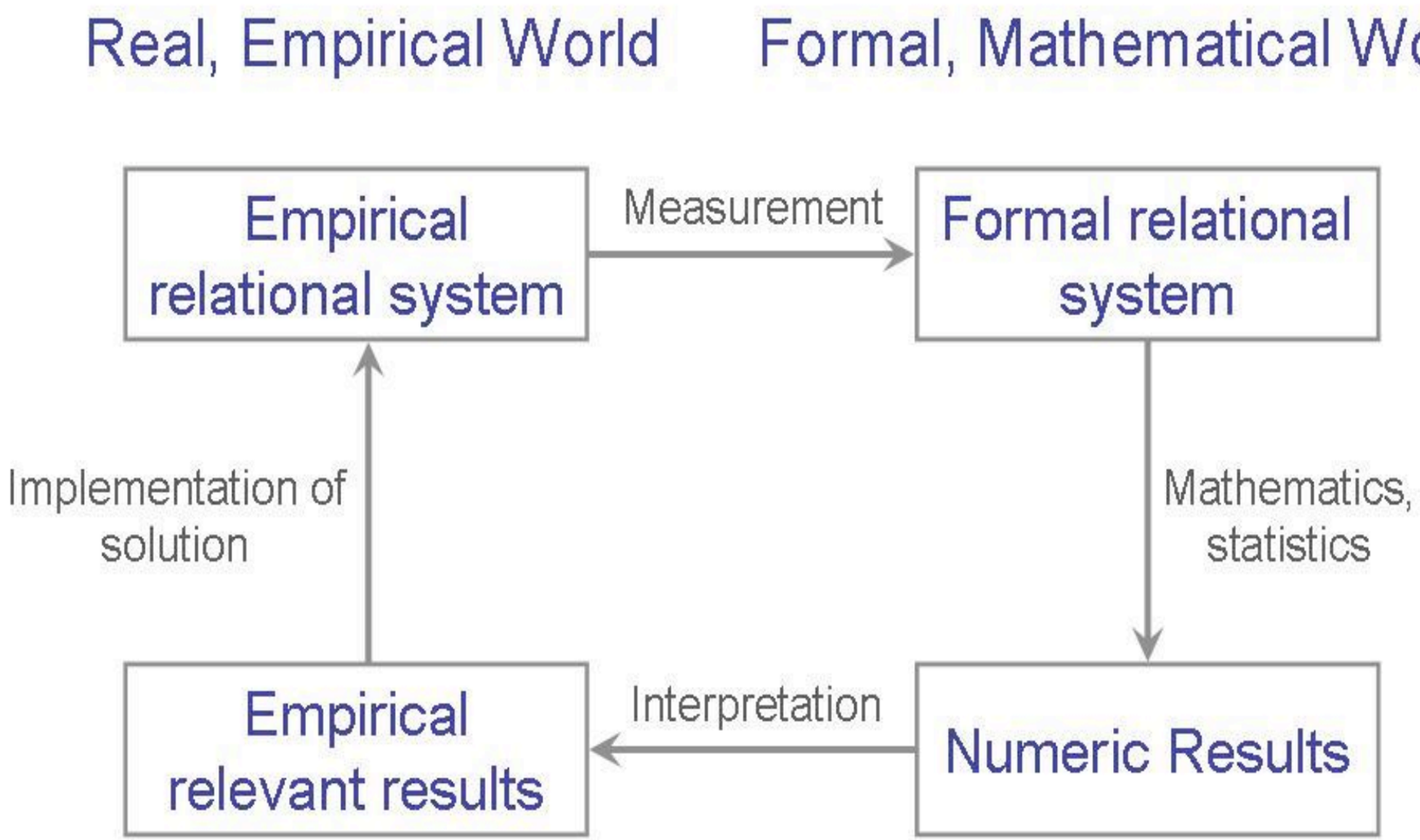
- How to structure the development process (as previously discussed)
- Many different types of process with many variations
    - Different types of software need different processes (no “one size fits all”)
- However, having a process gives us something to measure and analyze
    - We want to characterize the utility of a process with respect to a particular context (application domain, resources, skill of developers, etc.)

Software Reuse

- Commonalities between applications may allow reusing artifacts from previous developments
  - Improve productivity
  - Reduce costs
- Potential concerns
  - It may be faster to build a smaller application than searching for reusable components
  - Generalized components take more time to build
  - Must clarify who will be responsible for maintaining reusable components
  - Generality vs specificity: always a conflict

Measurement

- Using measurement to find or improve a software solution to a problem



Tools and Integrated Environments

- Platform integration (on heterogeneous networks)
- Presentation integration (commonality of user interface)
- Process integration (linking tools and the development process)
- Data integration (the ability to share data among disparate tools)
- Control integration (the ability of one tool to initiate action in another one)

Example Systems

- Book will make use of two example systems to illustrate SE techniques
  - Information System: Piccadilly Television Advertising System
    - Lots of complex rules that need to be followed
    - Large system with many inputs
  - Real-Time System: Ariane-5 flight control software
    - First Ariane-5 destroyed 40 seconds into maiden voyage
      - A software fault had caused the rocket's booster jets to be fired in random patterns causing the rocket to veer off course
    - 7 billion dollars worth of development went into construction of rocket system; 500 million dollars worth of satellites were on board

Wrapping Up

- Broad overview of Software Engineering
- Reviewed
    - Basic problem solving nature of the discipline
    - The notion of quality
    - The systems approach to software development
    - How SE has changed
    - Fundamental SE concepts

Coming Up Next

- Lecture 3: Introduction to Concurrency
    - Chapter 1 of Magee and Kramer