GRAND VALLEY STATE UNIVERSITY
www.gvsu.edu

Software Engineering Overview

Erik Fredericks // frederer@gvsu.edu

New aspect ratio how neat!

plus a git demo (stop me at 1:40)

Contents

Objectives

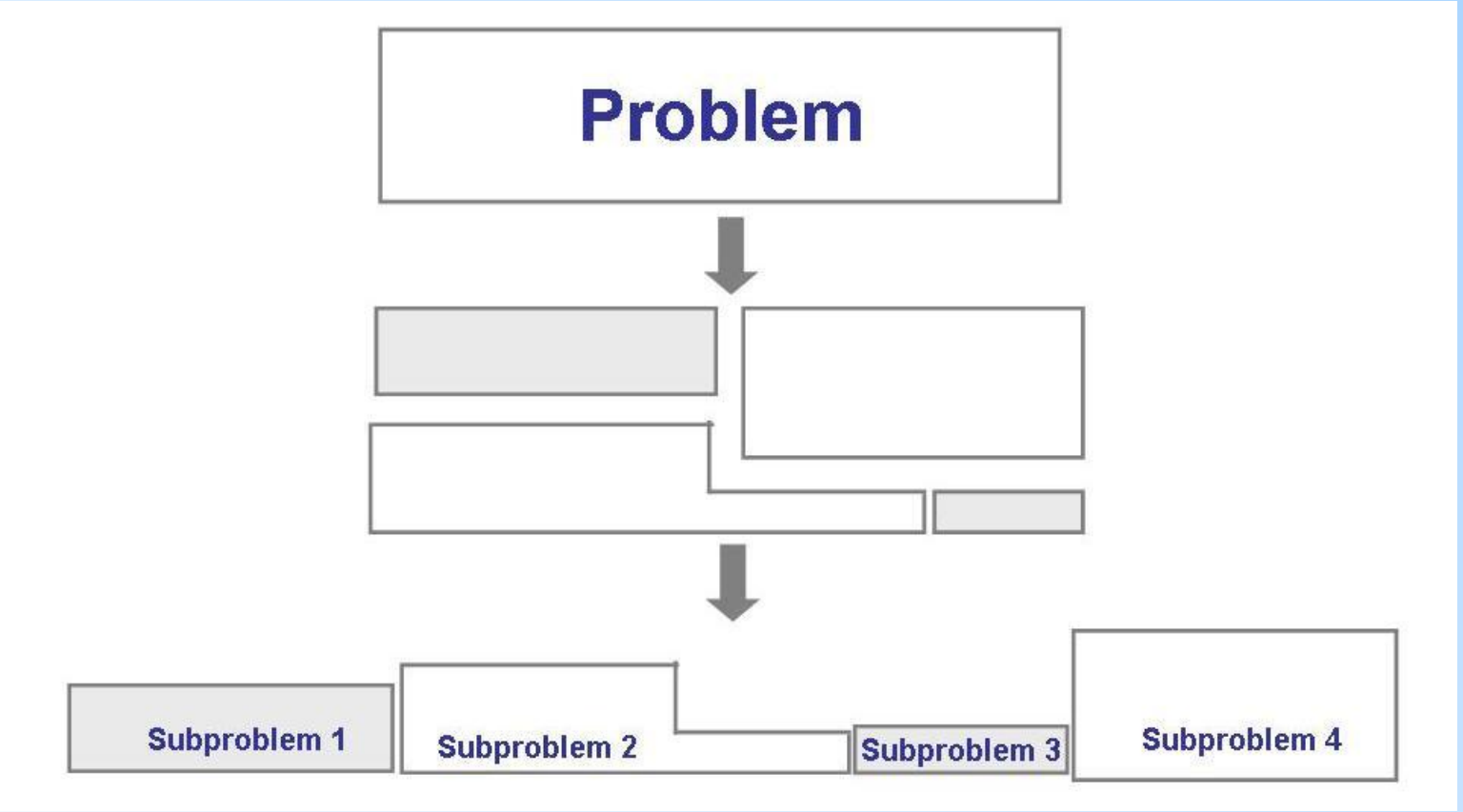
- What we mean by software engineering
- Software engineering's track record
- What we mean by good software
- Why a system approach is important
- How software engineering has changed since 1970s

1.1 What is Software Engineering
Solving Problems

- Software products are large and complex
- Development requires analysis and synthesis
  - *Analysis*: decompose a large problem into smaller, understandable pieces
    - abstraction is the key
  - *Synthesis*: build (compose) a software from smaller building blocks
    - composition is challenging

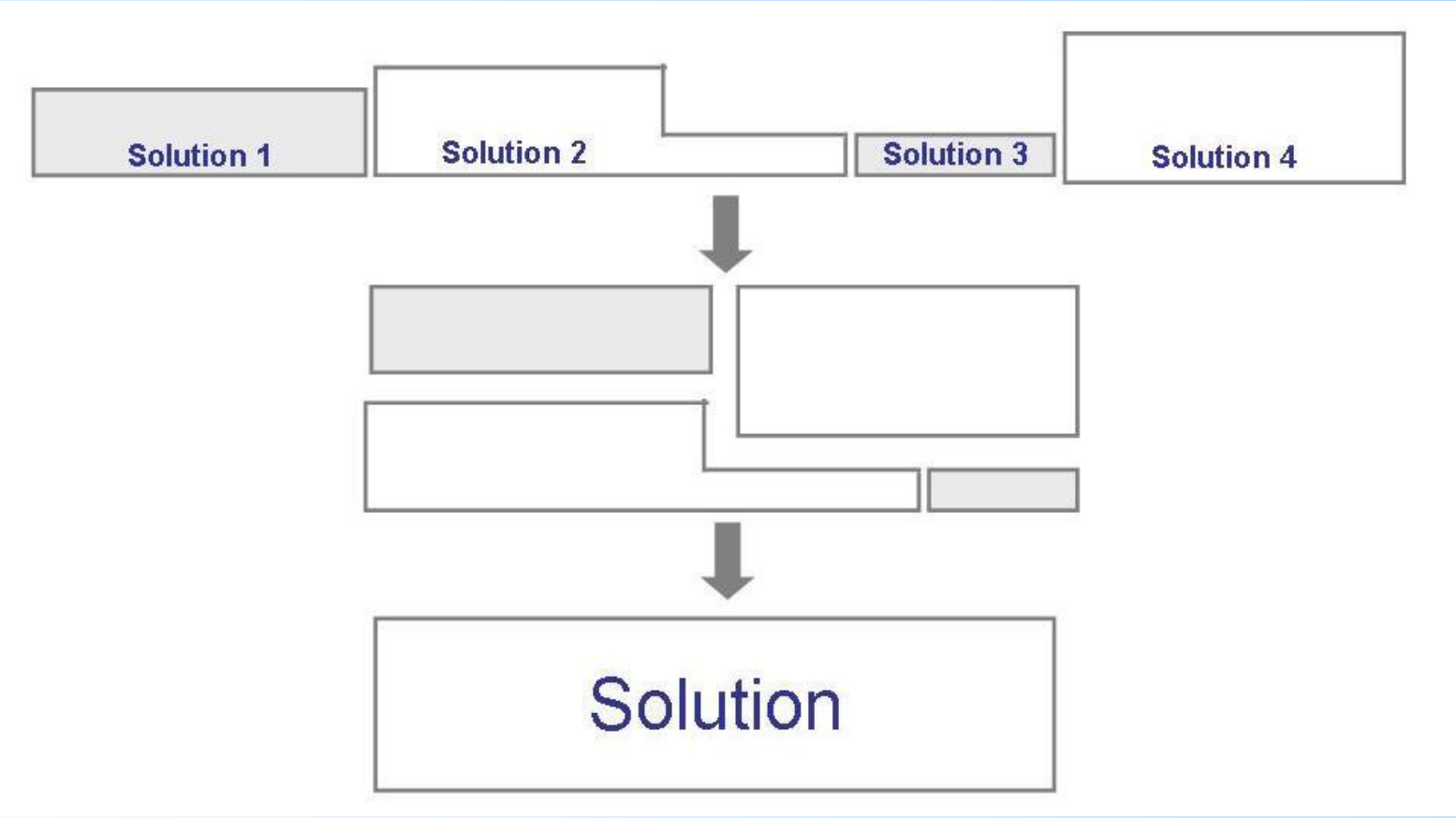
1.1 What is Software Engineering
Solving Problems (continued)

- The analysis process



1.1 What is Software Engineering
Solving Problems (continued)

- The synthesis process



1.1 What is Software Engineering
Solving Problems (continued)

- **Method**: refers to a formal procedure; a formal "recipe" for accomplishing a goal that is typically independent of the tools used
- **Tool**: an instrument or automated system for accomplishing something in a better way
- **Procedure**: a combination of tools and techniques to produce a product
- **Paradigm:** philosophy or approach for building a product (e.g., OO vs structured approaches)

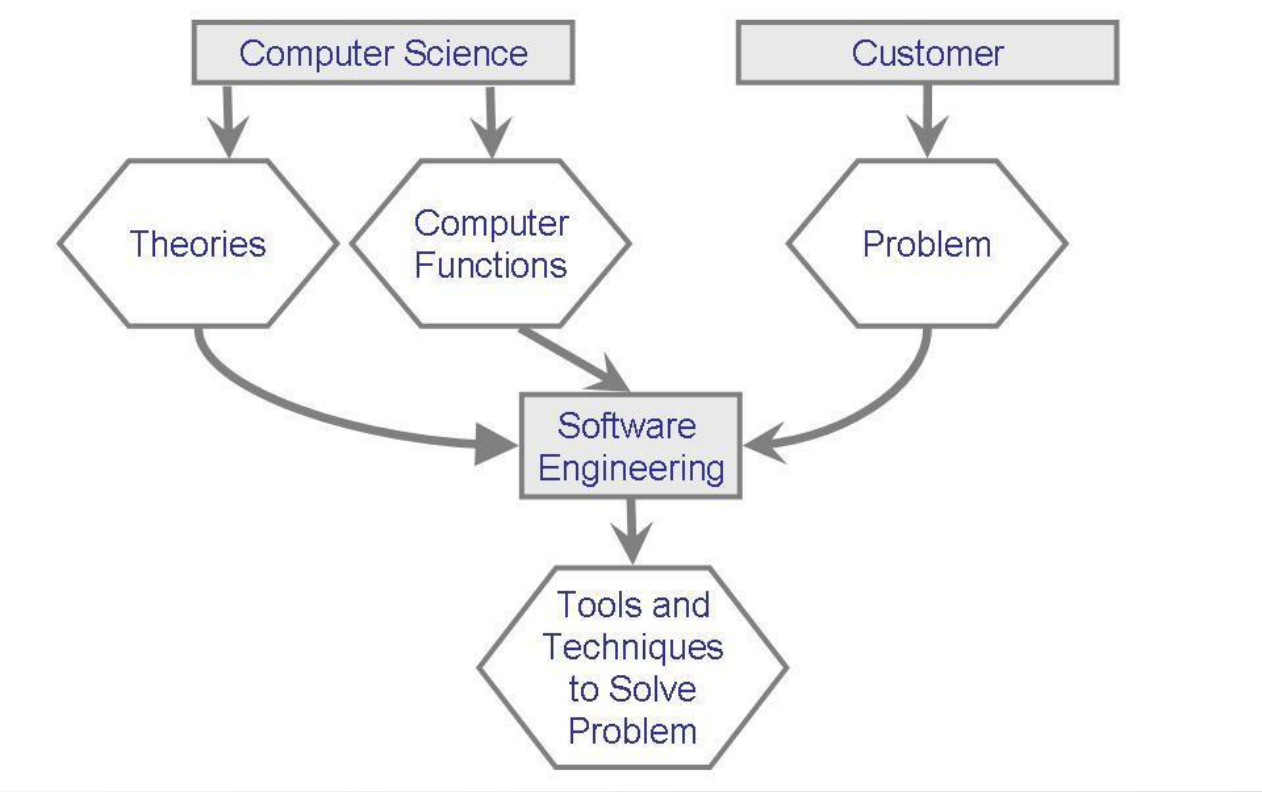
1.1 What is Software Engineering
Where Does the Software Engineer Fit In?

- **Computer science**: focusing on computer hardware, compilers, operating systems, and programming languages

- **Software engineering**: a discipline that uses computer and software technologies as a problem-solving tools

1.1 What is Software Engineering
Where Does the SW Engineer Fit in? (continued)

- Relationship between computer science and software engineering

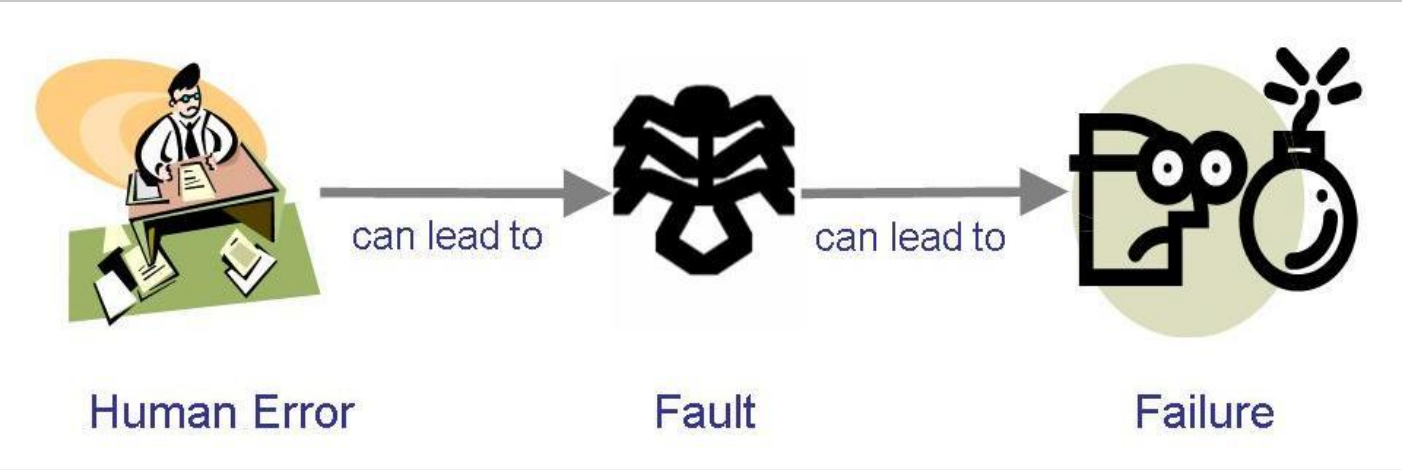


1.2 How Successful Have We Been?

- Perform tasks more quickly and effectively
  - Word processing, spreadsheets, e-mail
- Support advances in medicine, agriculture, transportation, multimedia education, and most other industries
- Many good stories
- However, software is not without problems

1.2 How Successful Have We Been?
Sidebar 1.1 Terminology for Describing Bugs

- **A fault**: occurs when a human makes a mistake, called **an error**, in performing some software activities
- **A failure**: is a departure from the system's required behaviour



1.2 How Successful Have We Been?
Examples of Software Failure

- 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
- Malfunctioning code in Therac-25 killed several people
- Reliability constraints have caused cancellation of many *safety critical* systems
  - *Safety-critical*: something whose failure poses a threat to life or health

1.3 What Is Good Software?
Sidebar 1.2 Perspective on Quality

- The **transcendental** view: quality is something we can recognize but not define
- The **user** view: quality is fitness for purpose
- The **manufacturing** view: quality is conformance to specification
- The **product** view: quality tied to inherent product characteristics
- The **value-based** view: depends on the amount the customers is willing to pay for it

1.3 What is Good Software?

- Good software engineering must always include a strategy for producing quality software
- Three ways of considering quality
  - The quality of the product
  - The quality of the process
  - The quality of the product in the context of the business environment

1.3 What Is Good Software?
The Quality of the Product

- Users judge external characteristics (e.g., correct functionality, number of failures, type of failures)
- Designers and maintainers judge internal characteristics (e.g., types of faults)
- Thus different stakeholders may have different criteria
- Need quality models to relate the user's external view to developer's internal view


ARASAKA
ACCESS SUSPENDED // SYSTEM SHUTDOWN IN PROGRESS //
ARASAKA AGENT
F
[Surrender shard] It's all here.
How'd you find out?
Arasaka Agent: The data. Now.

1.3 What Is Good Software?
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
  - Where to find a particular kind of fault
  - How to find faults early
  - How to build in fault tolerance
  - What are alternative activities

1.3 What Is Good Software?
The Quality of the Process (continued)

- Models for process improvement
  - SEI's Capability Maturity Model (CMM)
  - ISO 9000
  - Software Process Improvement and Capability dEtermination (SPICE)

1.3 What Is Good Software?

The Quality in the Context of the Business Environment

- Business value is as important as technical value
- Business value (in relationship to technical value) must be quantified
- A common approach: **return on investment (ROI)** – what is given up for other purposes
- ROI is interpreted in different terms: reducing costs, predicting savings, improving productivity, and costs (efforts and resources)

What do you think the customer's role is within a project?

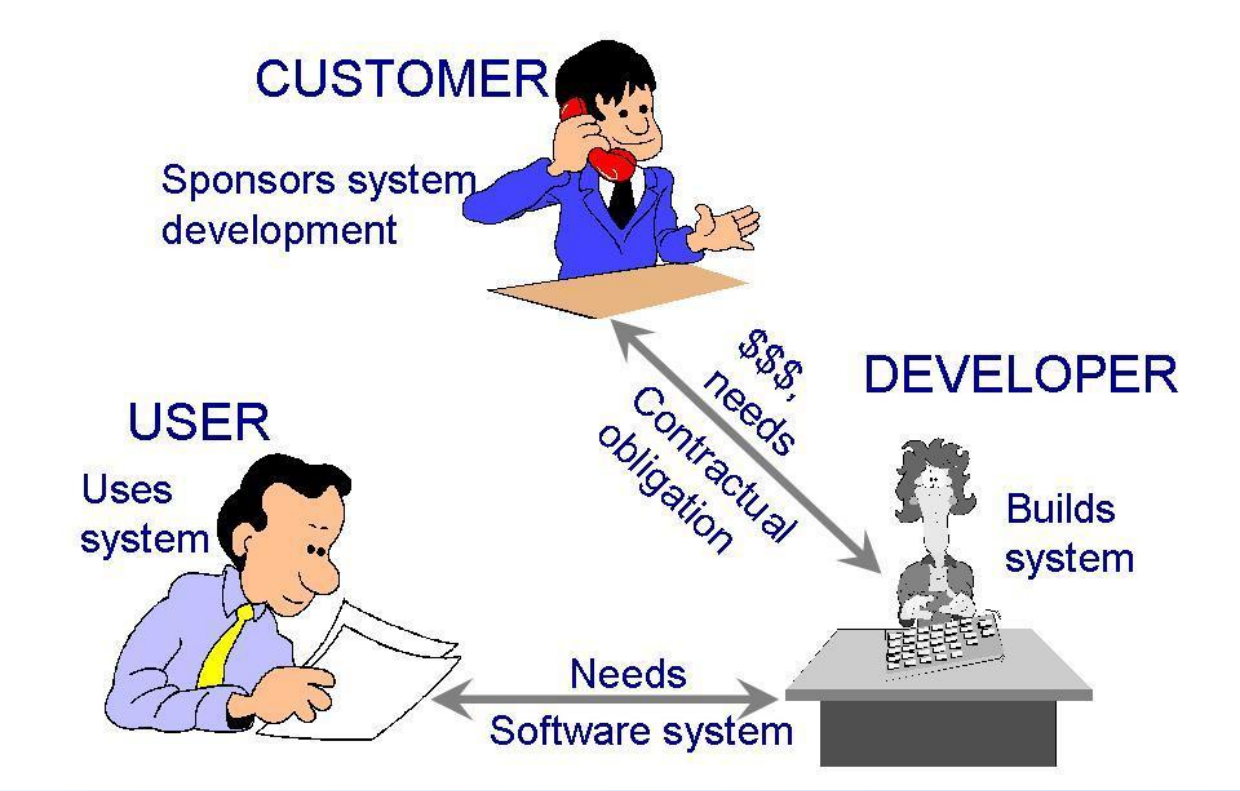
~~**What do you think a STAKEHOLDER is?**~~

1.4 Who Does Software Engineering?

- **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

1.4 Who Does Software Engineering? (continued)

- Participants (stakeholders) in a software development project



1.5 System Approach

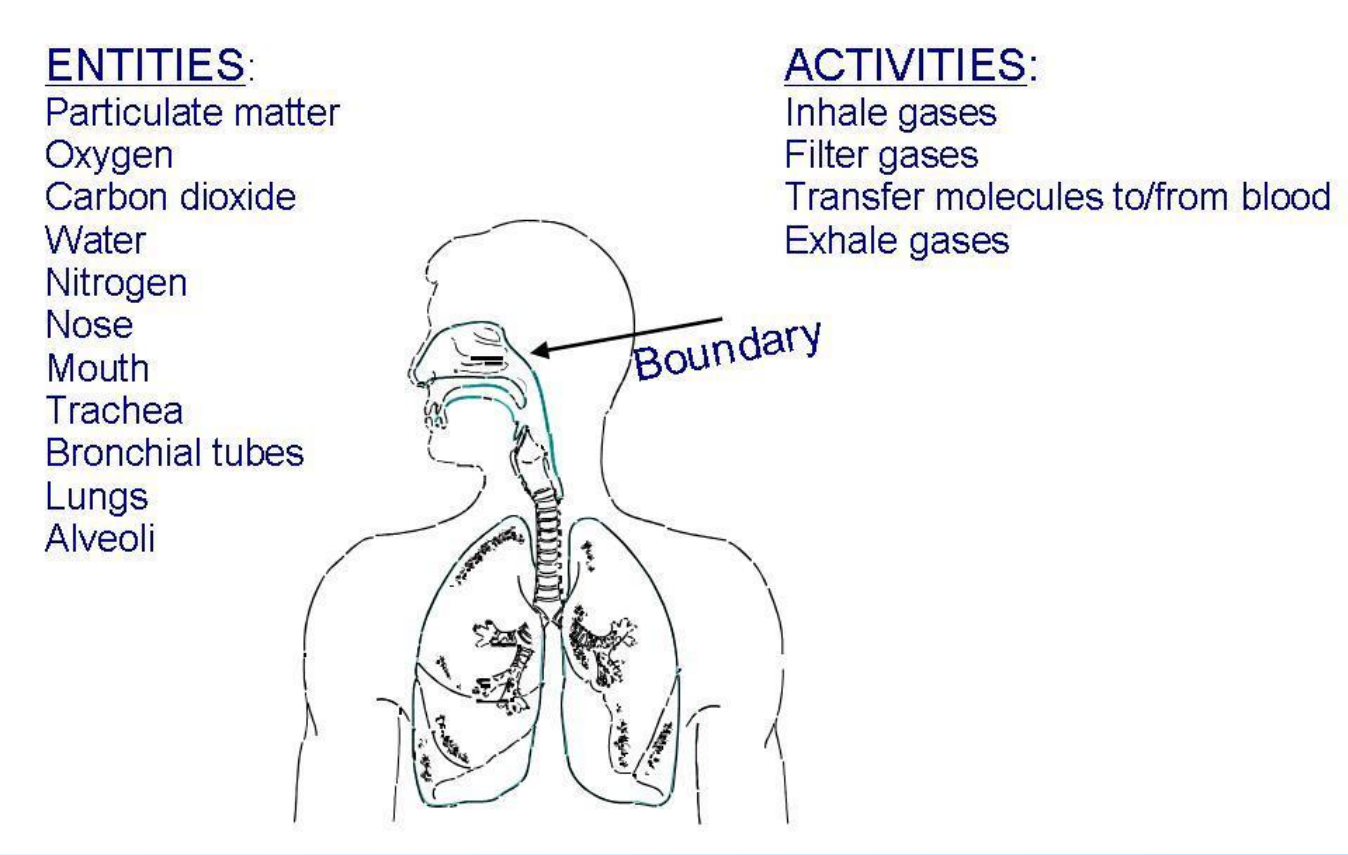
- Hardware, software, interaction with people
- Identify activities and objects
- Define the system boundary
- Consider nested systems, systems interrelationship

1.5 System Approach
The Element of a System

- 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

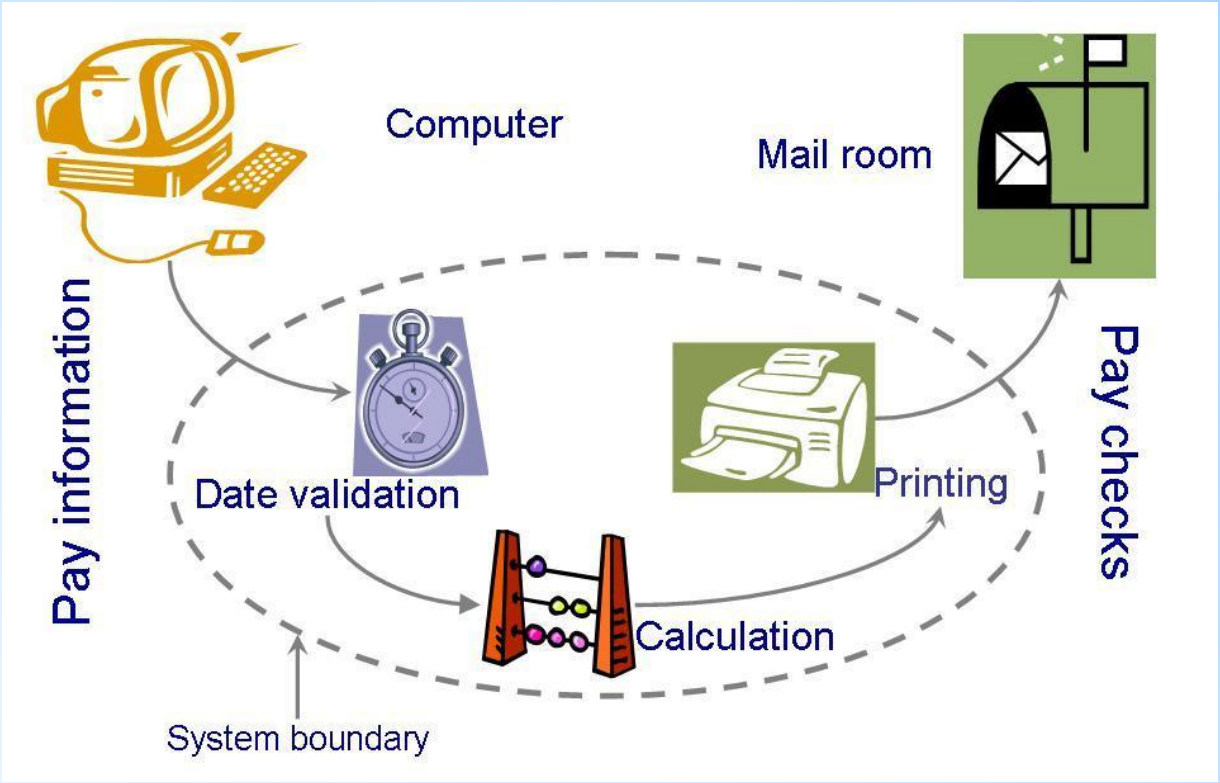
1.5 System Approach
The Element of a System (continued)

- Example of systems: a human respiratory system



1.5 System Approach
The Element of a System (continued)

- A computer system must also be clearly described: System definition of a paycheck production

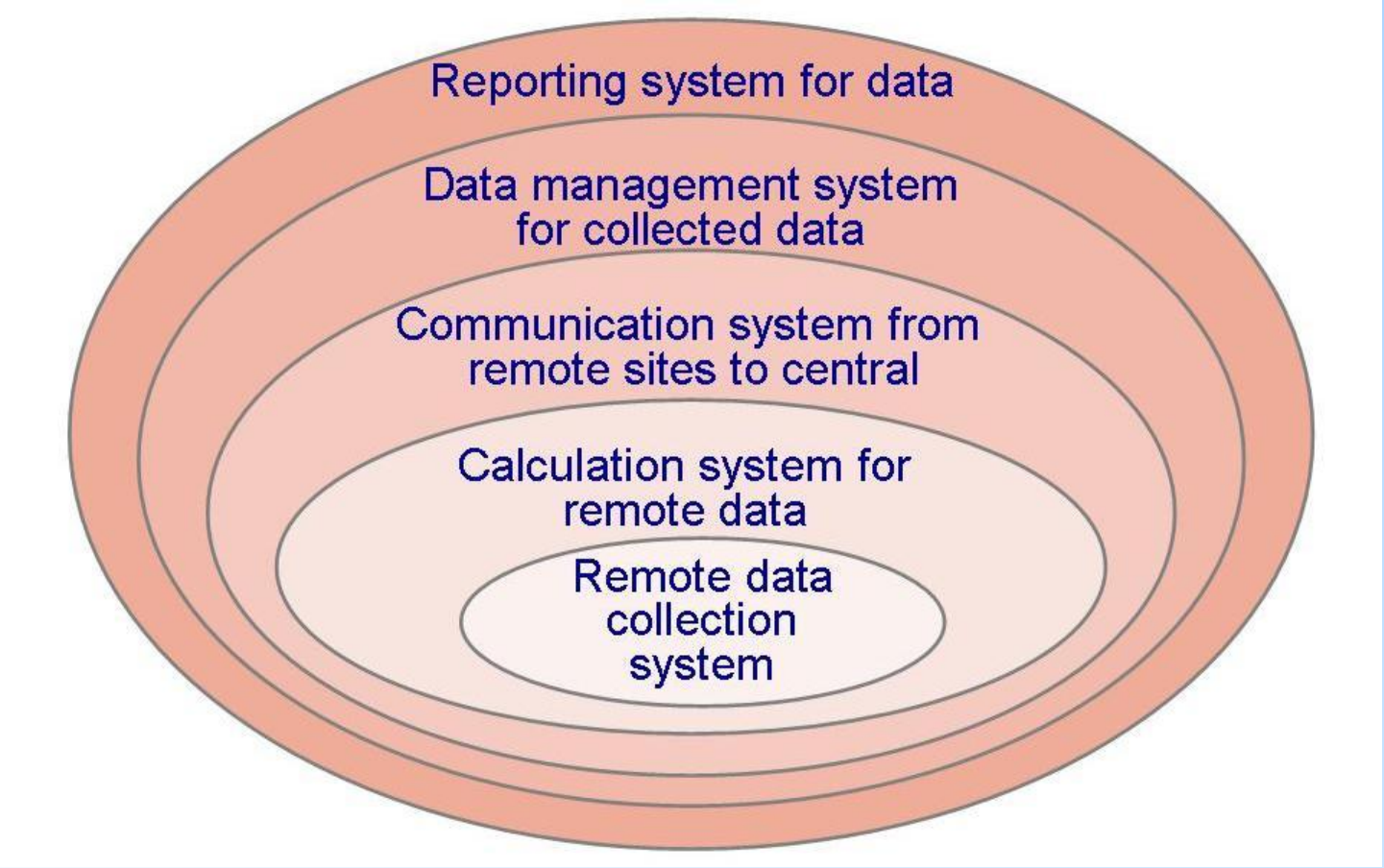


1.5 System Approach
Interrelated Systems

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

1.5 System Approach
Interrelated Systems (continued)

- A layered system



1.6 Engineering Approach
Building a System

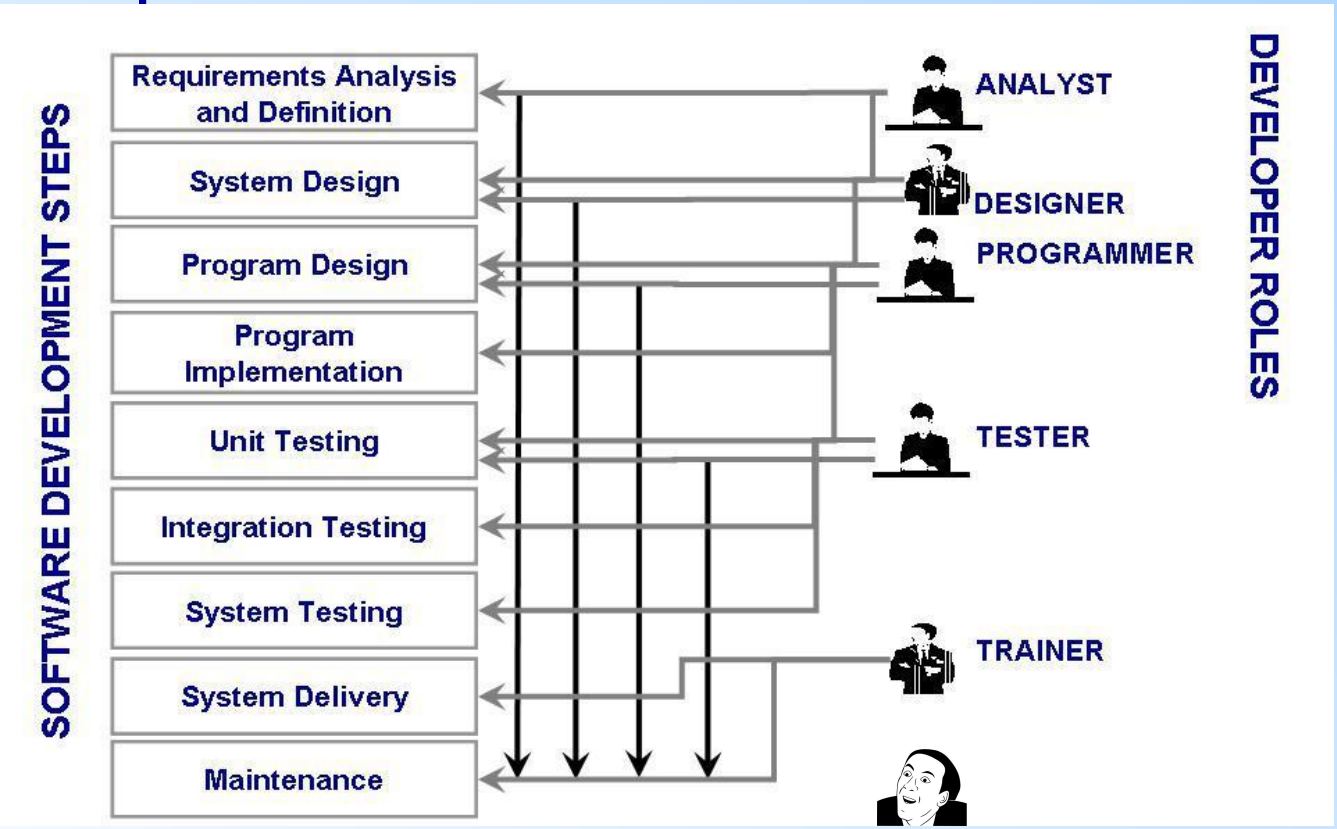
- Requirement analysis and definition
- System design
- Program design
- Writing the programs
- Unit testing
- Integration testing
- System testing
- System delivery
- Maintenance

1.7 Members of the Development Team

- **Requirement analysts**: work with the customers to identify and document the requirements
- **Designers**: generate a system-level description of what the system is 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

1.7 Members of the Development Team (continued)

- Typical roles played by the members of a development team



1.8 How Has Software Engineering Changed?
The Nature of the Change

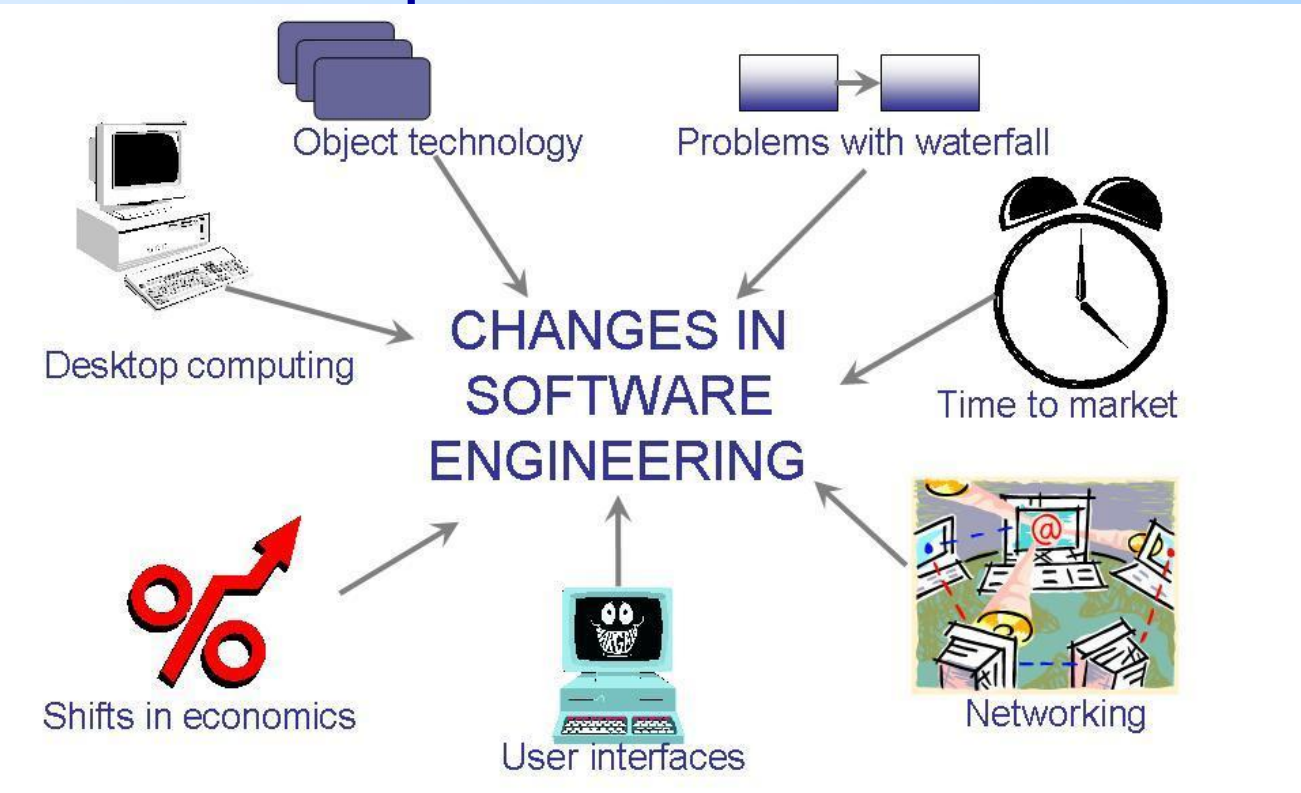
- Before 1970s
  - Single processors: mainframes
  - Designed in one of two ways
    - as a **transformation**: input was converted to output
    - as a **transaction**: input determined which function should be performed
- After 1970s
  - Run on multiple systems
  - Perform multi-functions

1.8 How Has SE Changed?
Wasserman's Seven Key Factors

1. Critically of time-to-market
2. Shifts in the economics of computing
3. Availability of powerful desktop computing
4. Extensive local- and wide-area networking
5. Availability and adoption of object-oriented technology
6. Graphical user interfaces
7. Unpredictability of the waterfall model of software development

1.8 How Has SE Changed?
Wasserman's Seven Key Factors (continued)

- The key factors that have changed the software development

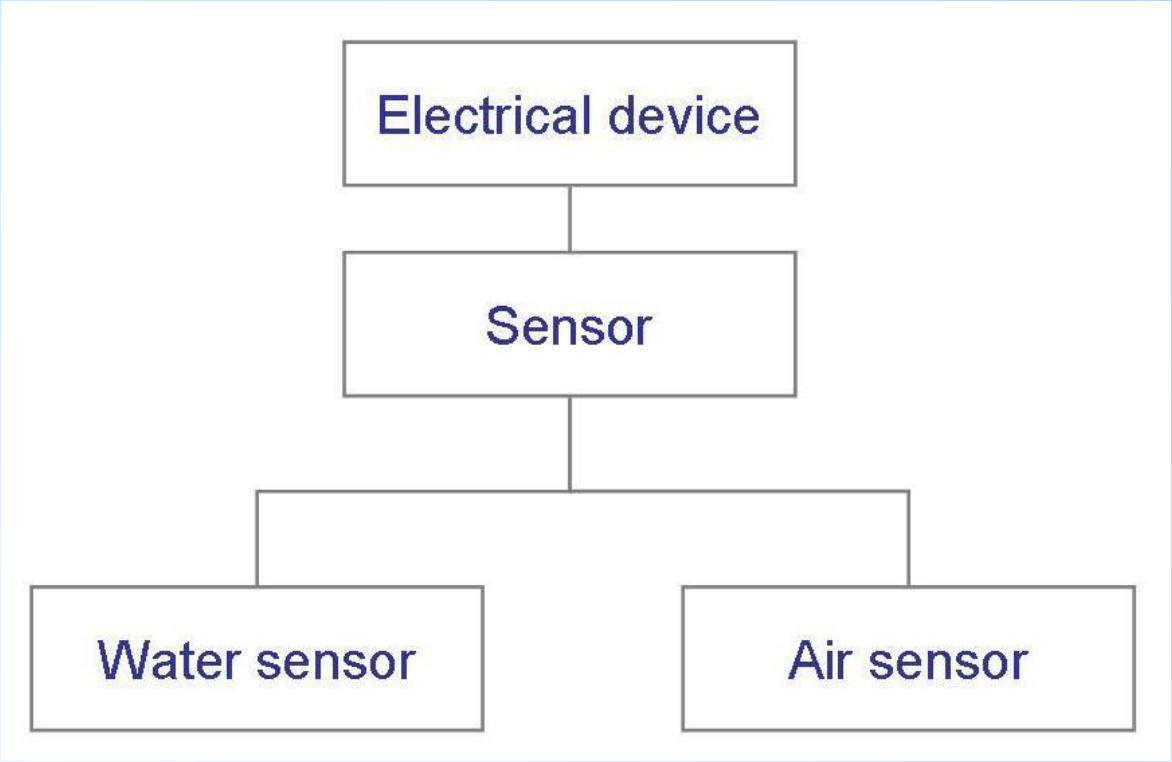


1.8 How Has SE Changed?
Wasserman's Discipline of Software Engineering

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

1.8 How Has SE Changed?
Abstraction

- A description of the problem at some level of generalization
  - Hide details



1.8 How Has SE Changed?
Analysis and Design Methods and Notations

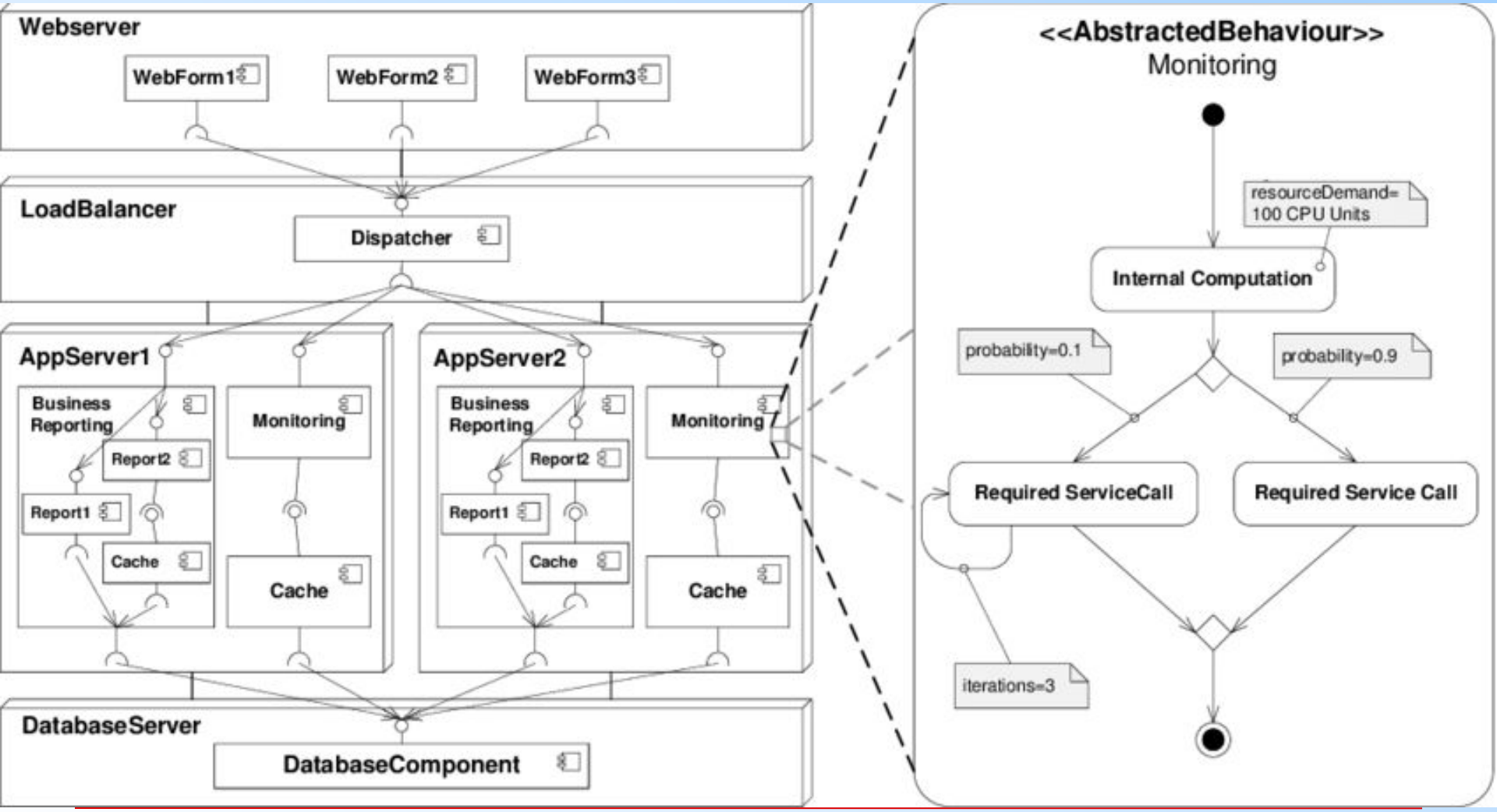
- Provide documentation
- Facilitate communication
- Offer multiple views
- Unify different views

1.8 How Has SE Changed?
User Interface Prototyping

- Prototyping: building a small version of a system
  - Help users identify key requirements of a system
  - Demonstrate feasibility
- Develop good user interface

1.8 How Has SE Changed?
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
  - Object-oriented decomposition

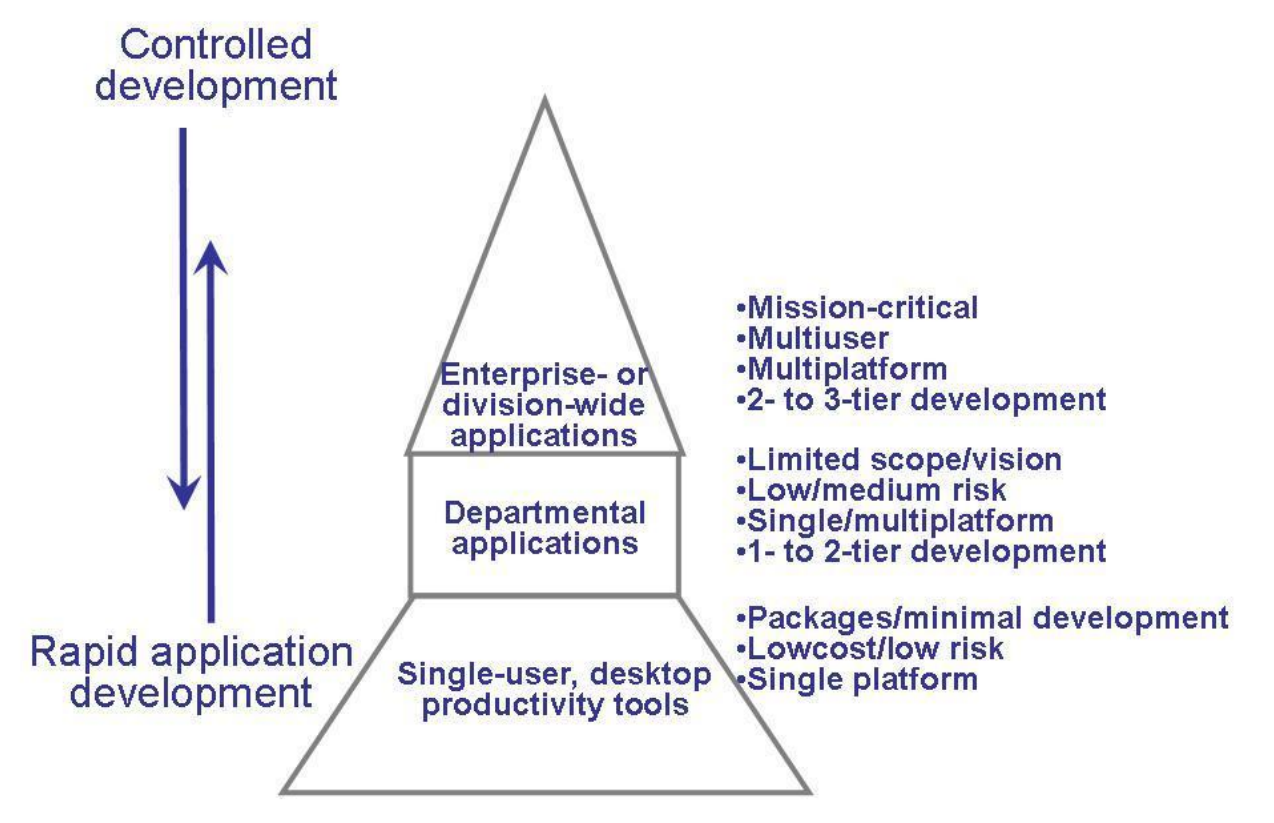

Webserver
WebForm1
WebForm2
WebForm3
LoadBalancer
Dispatcher
AppServer1
Business Reporting
Report2
Report1
Cache
Monitoring
Cache
AppServer2
Business Reporting
Report2
Report1
Cache
Monitoring
Cache
DatabaseServer
DatabaseComponent
<<AbstractedBehaviour>>
Monitoring
resourceDemand=
100 CPU Units
Internal Computation
probability=0.1
probability=0.9
Required ServiceCall
Required Service Call
iterations=3

1.8 How Has SE Changed?
Software Process

- Many variations
- Different types of software need different processes
  - Enterprise-wide applications need a great deal of control
  - Departmental applications can take advantage of rapid development

1.8 How Has SE Changed?
Software Process (continued)

- Pictorial representation of differences in development processes

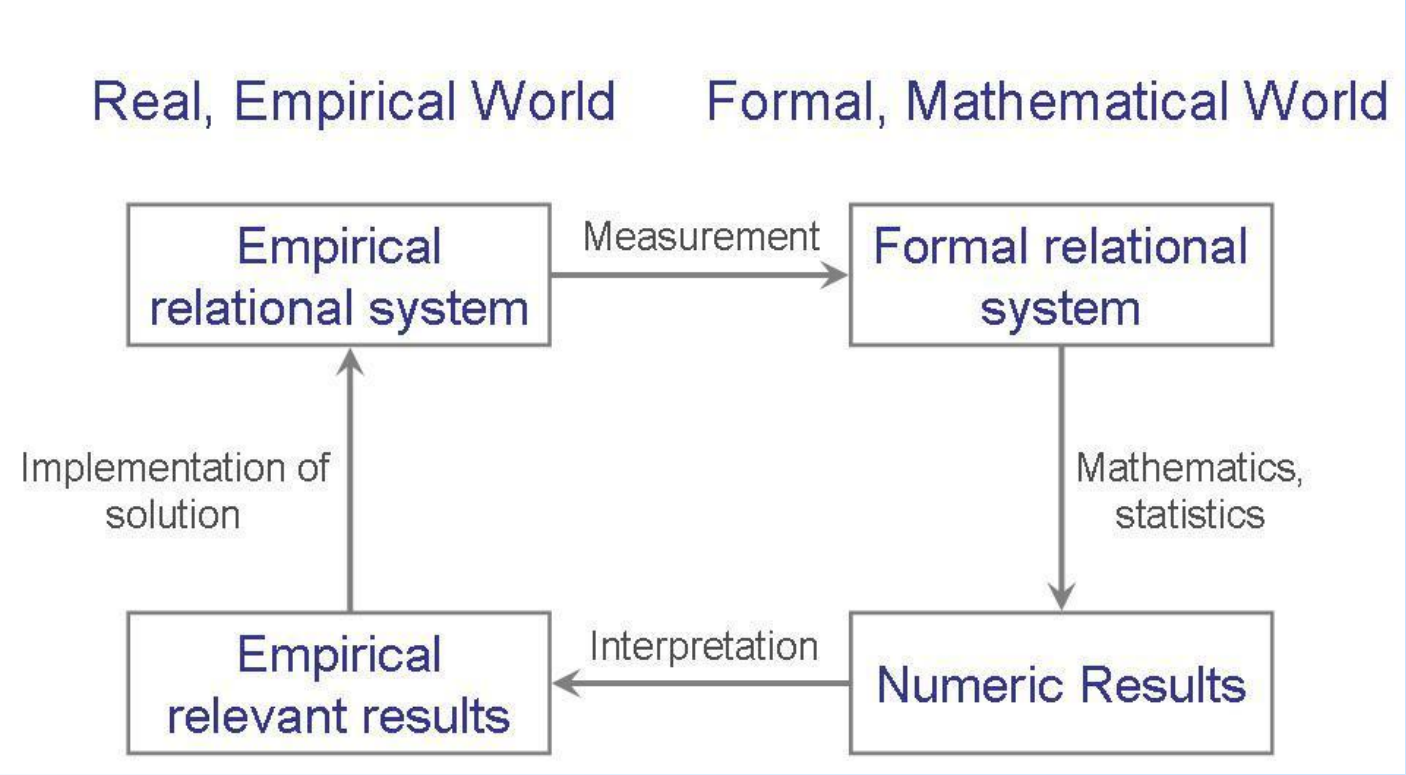


1.8 How Has SE Changed?
Software Reuse

- 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

1.8 How Has SE Changed?
Measurement

- Objective: describe quality goals in a quantitative way



1.8 How Has SE Changed?
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** (to share common data)
- **Control integration** (the ability of one tool to initiate action in another one)

1.10 Real Time Example

- Ariane-5 rocket, from the European Space Agency
- June 4, 1996: functioned well for 40 seconds, then veered off course and was destroyed
- Contained four satellites: cost was $500 million
- Reused code from Ariane-4 rocket

1.10 Real Time Example
Ariane-5 Definition of Quality

- From the Lions et al report:
  - "… demonstrated the high quality of the Ariane-5 programme as regards engineering work in general and completeness and traceability of documents."
  - "… the supplier of the SRI … was only following the specification given to it. … The exception which occurred was not due to random failure but a design error."

1.11 What this Means for You

- Given a problem to solve
  - Analyze it
  - Synthesize a solution
- Understand that requirements may change
- Must view quality from several different perspectives
- Use fundamental software engineering concepts (e.g., abstractions and measurements)
- Keep system boundary in mind

What is Next?

Software Process Models:
Organizing Software Development