SOFTWARE and SOFTWARE ENGINEERING

- The Nature of Software
- History of Software Development
- Software Engineering Paradigms and Technology
- Software Complexity, Object-Oriented Requirements Analysis (OORA), and Object-Oriented Design (OOD)

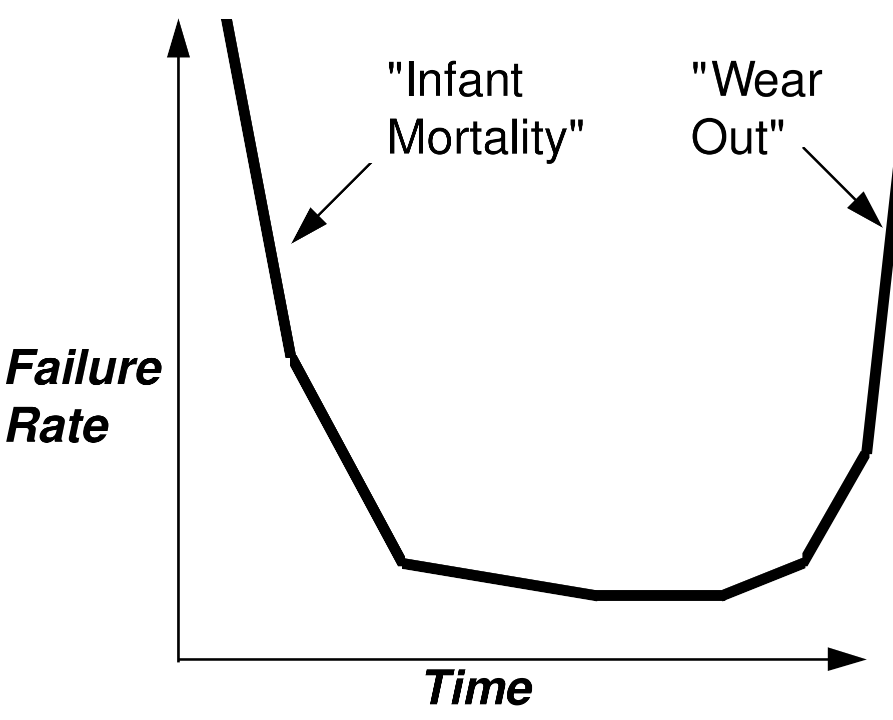
THE NATURE OF SOFTWARE

- ✓ Characteristics of Software
- ✓ Failure Curves for Hardware and Software
- ✓ Software Components
- ✓ Software Configuration

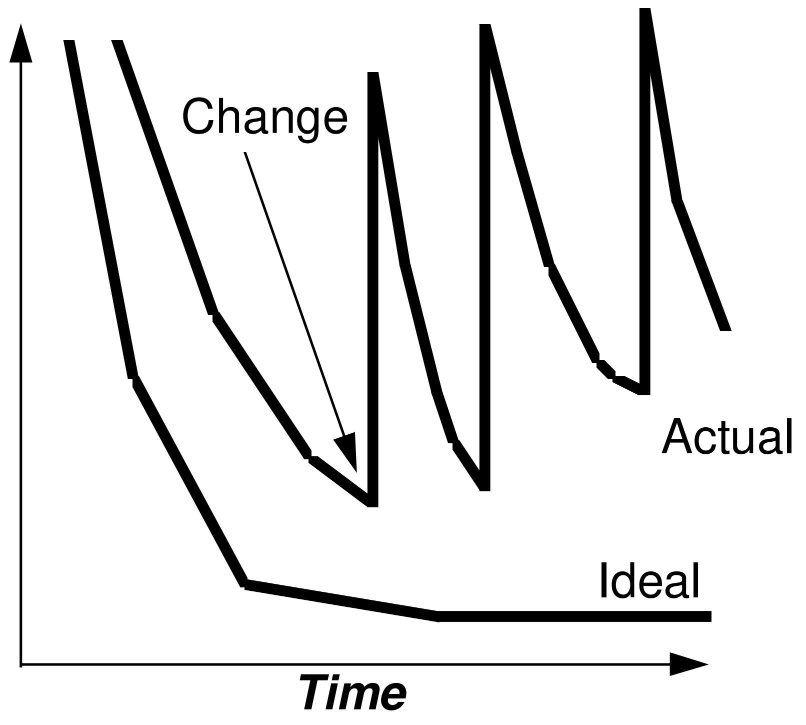
Characteristics of Software

- Software is *programs*, *documents*, and *data*.
- Software is developed or engineered; it is not manufactured like hardware.
- Software does not wear out, but it does *deteriorate*.
- Most software is custom-built, rather than being assembled from existing components.
- Software is a *business opportunity*.

Failure Curves for Hardware and Software



FAILURE CURVE FOR HARDWARE

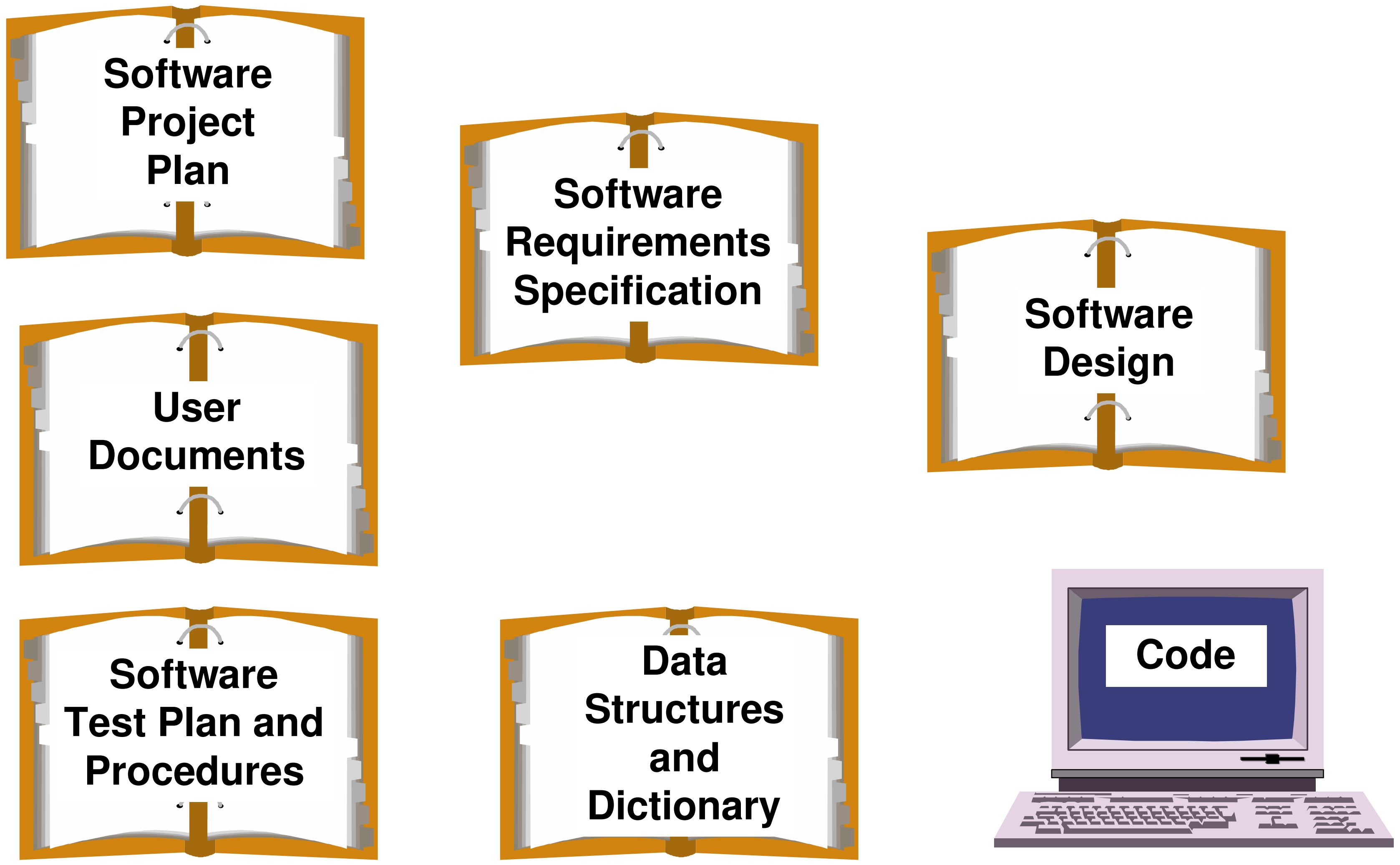


FAILURE CURVE FOR SOFTWARE

Software Components

- Software programs, or software systems, consist of *components*.
- A set of components which comprise a logical unit of software is called a *software configuration item*.
- Reuse and development of reliable, trusted software components improves software *quality* and *productivity*.
- Computer language forms:
  - Machine level (microcode, digital signal generators)
  - Assembly language (PC assembler, controllers)
  - High-order languages (FORTRAN, Pascal, C, Ada, ...)
  - Specialized languages (LISP, OPS5, Prolog, ...)
  - Fourth generation languages (databases, windows apps)

Software Configuration


Software Project Plan
Software Requirements Specification
Software Design
User Documents
Software Test Plan and Procedures
Data Structures and Dictionary
Code

Software Configuration

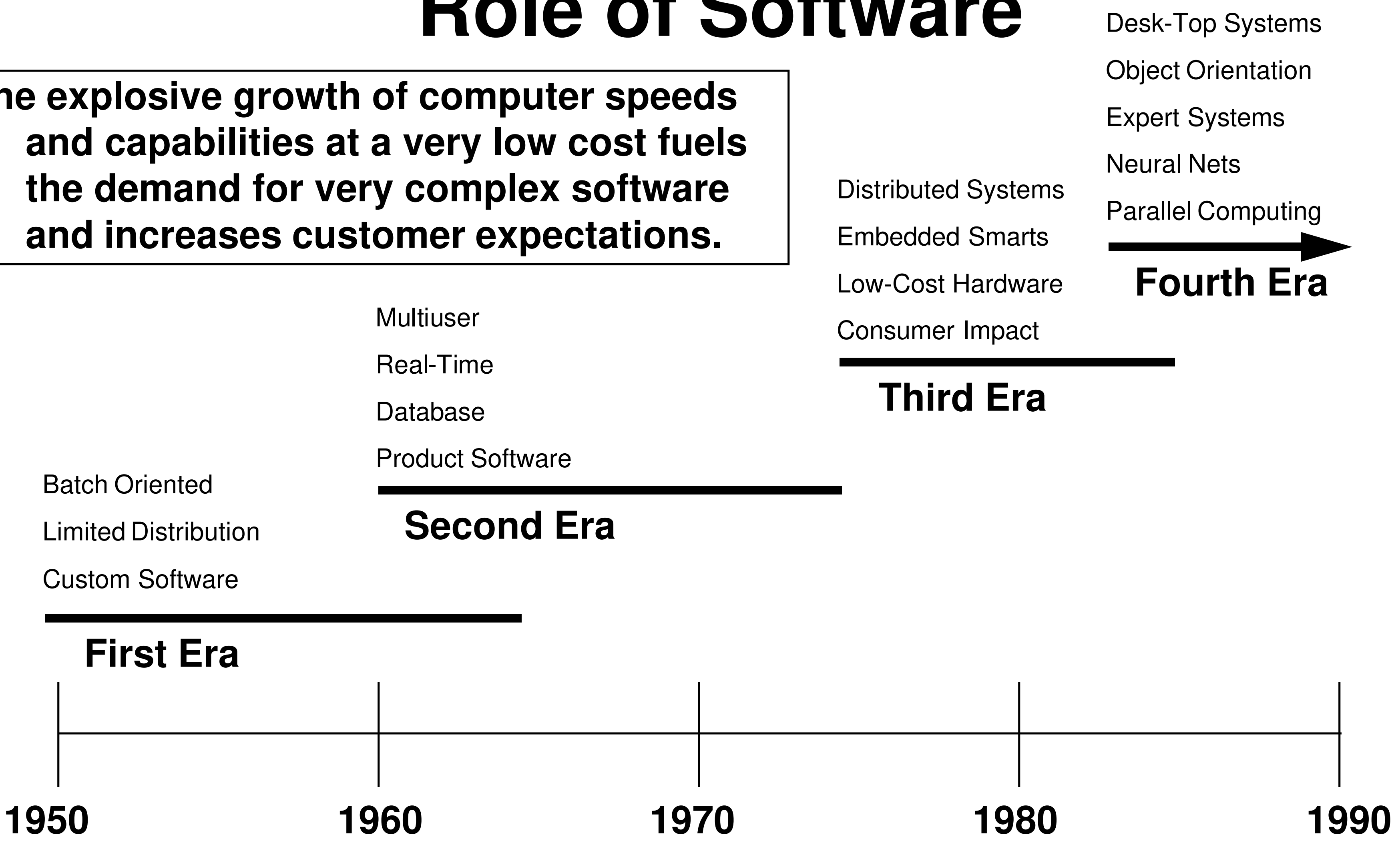
- Planning Activity
  - Software Project Plan
- Requirements Definition Activity
  - Software Requirements Specification
  - Software Test Plan and Procedures
  - Data Structures and Dictionary
  - User Documents
- Design Activity
  - Software Design Documents
  - Software Test Plan and Procedures
  - Data Structures and Dictionary
- Coding and Testing Activity
  - Code
  - Software Test Plan and Procedures
- Delivery and Maintenance Activity
  - User Documents
  - Others as needed

HISTORY OF SOFTWARE DEVELOPMENT

- ✓ Role of Software
- ✓ Industrial View

Role of Software

The explosive growth of computer speeds and capabilities at a very low cost fuels the demand for very complex software and increases customer expectations.



Industrial View

- Why does it take so long to finish a working software system?
- Why are development costs so high?
- Why can't we find all software errors before software is delivered?
- How can we measure the progress of software development?
- How can we survive in the global economy?

SOFTWARE ENGINEERING PARADIGMS

- ✓ What is Software Engineering?
- ✓ Life Cycle
- ✓ Prototyping Model
- ✓ Spiral Model
- ✓ Software Engineering Capability

What Is Software Engineering?

Methods

- Analysis
- Design
- Coding
- Testing
- Maintenance

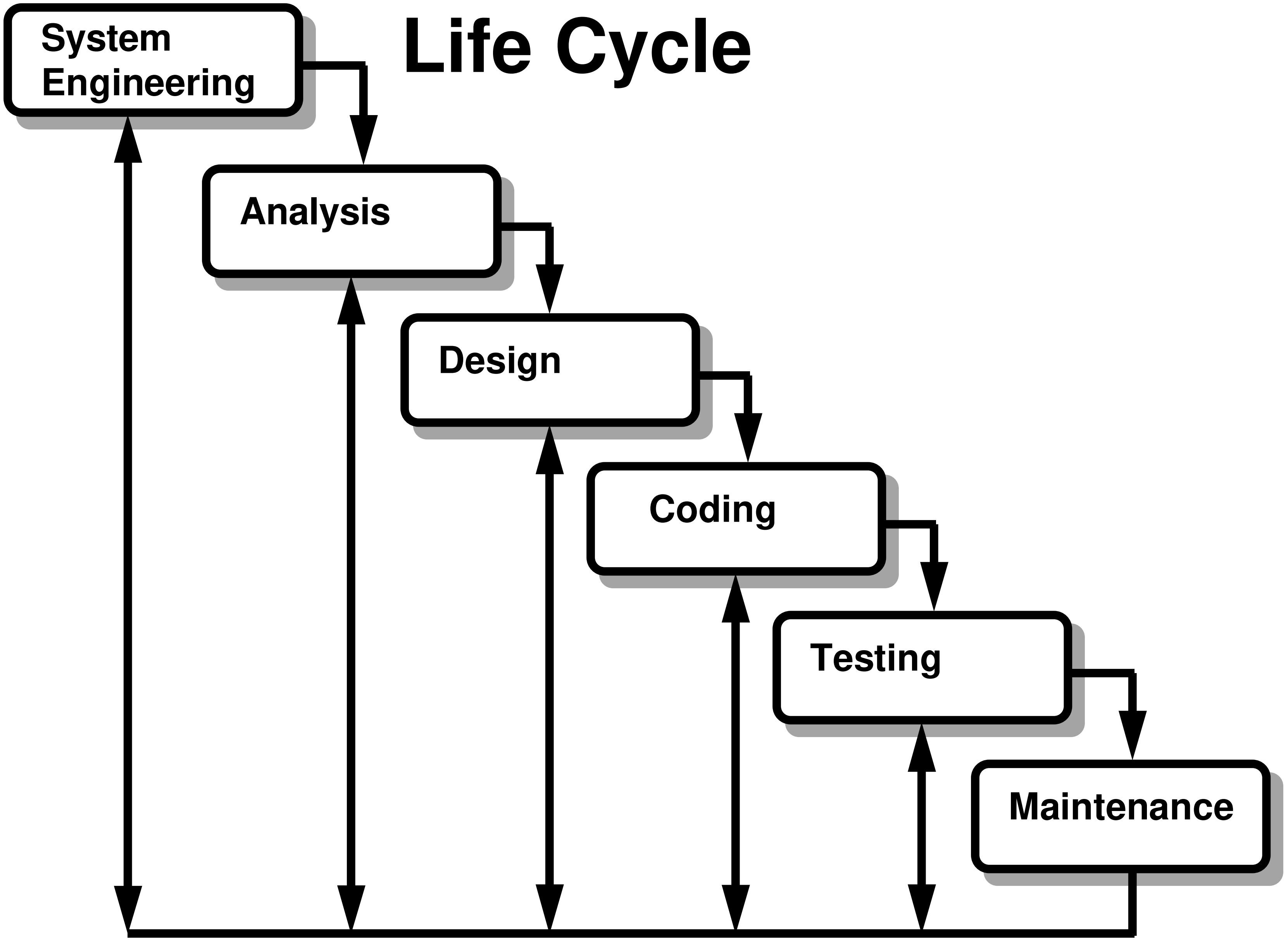
Procedures

- Project Management
- Software Quality Assurance
- Software Configuration Management
- Measurement
- Tracking
- Innovative Technology Insertion

Computer-Aided Software Engineering **(CASE)**

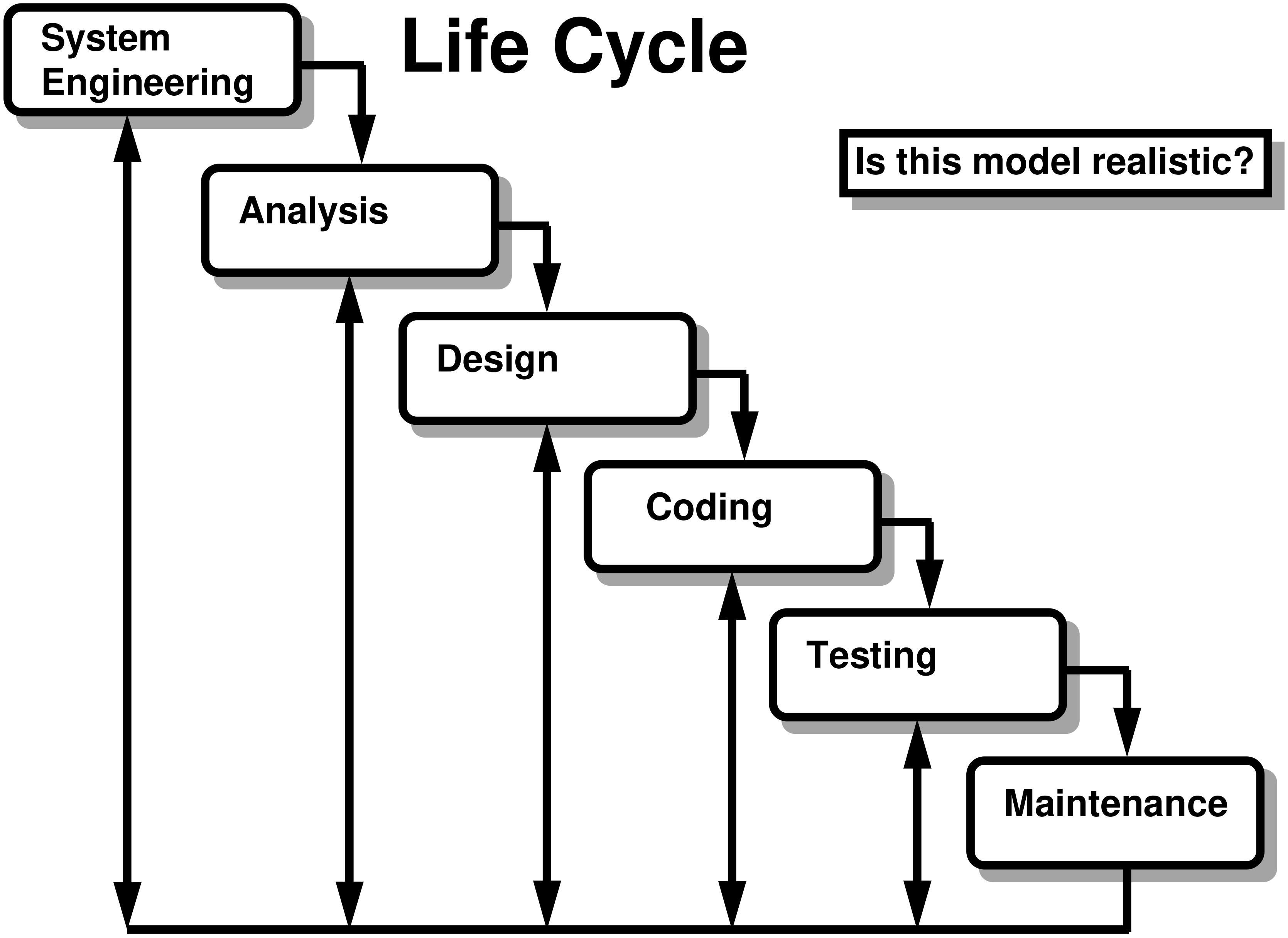
- **Tools which support the *Methods* and *Procedures***

Life Cycle

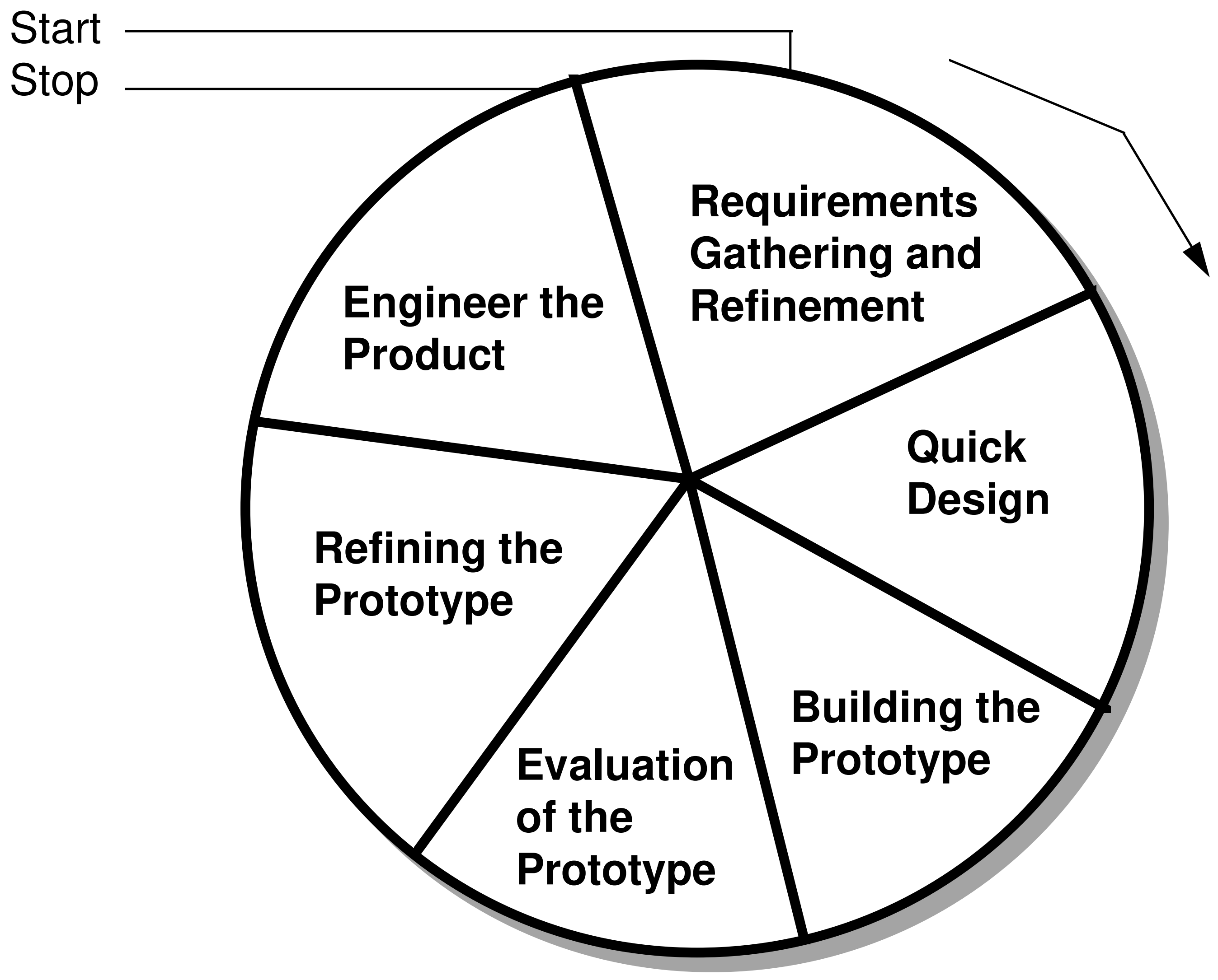

System Engineering
Analysis
Design
Coding
Testing
Maintenance

Life Cycle

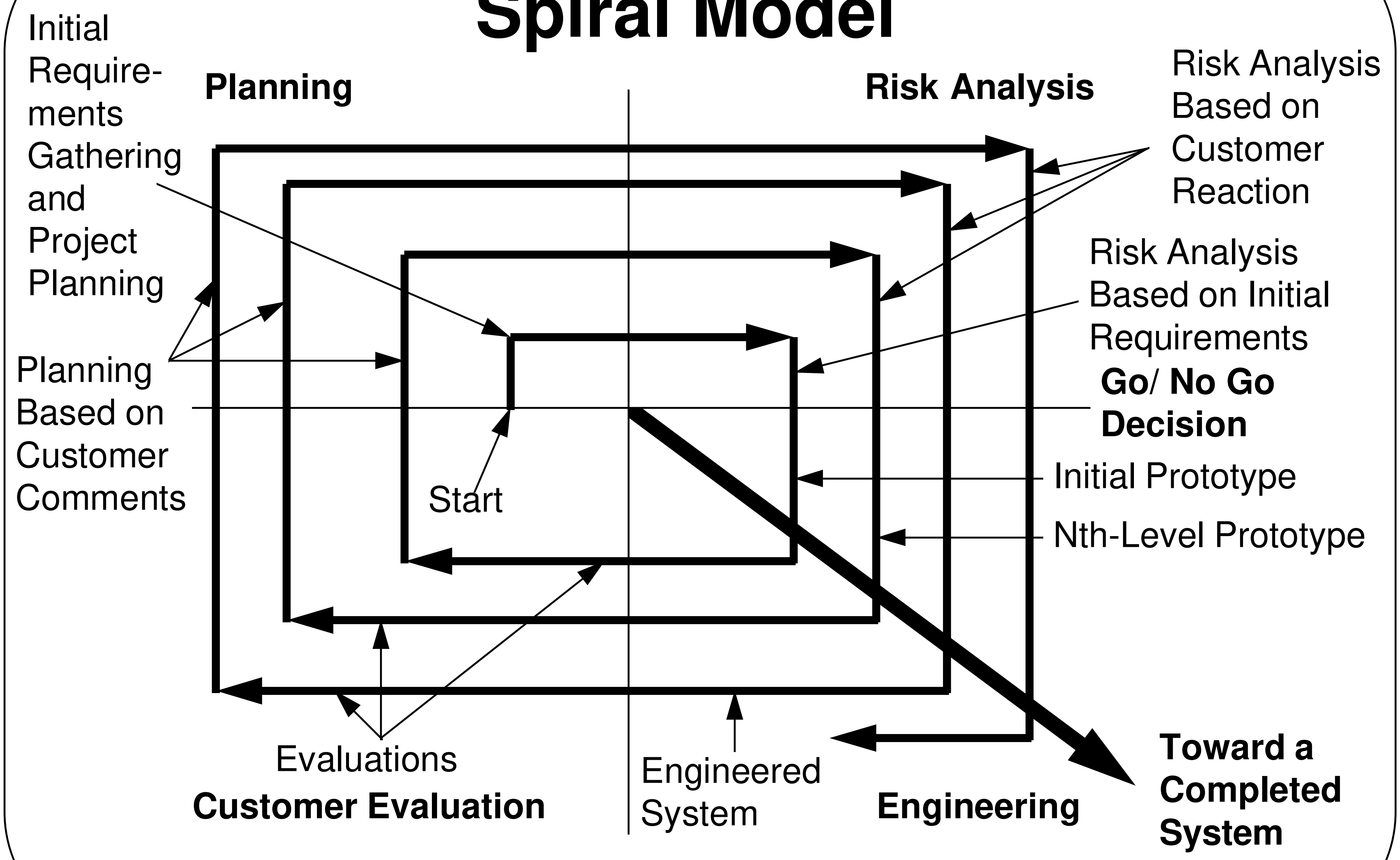
Is this model realistic?



Prototyping Model



Spiral Model


Initial Requirements Gathering and Project Planning
Planning
Risk Analysis
Risk Analysis Based on Customer Reaction
Risk Analysis Based on Initial Requirements
Go/ No Go Decision
Planning Based on Customer Comments
Start
Initial Prototype
Nth-Level Prototype
Evaluations
Customer Evaluation
Engineered System
Engineering
Toward a Completed System

Generic Paradigm

1. DEFINITION PHASE
   - System Analysis
   - Software Project Planning
   - Requirements Analysis

---

2. DEVELOPMENT PHASE
   - Software Design
   - Coding
   - Software Testing

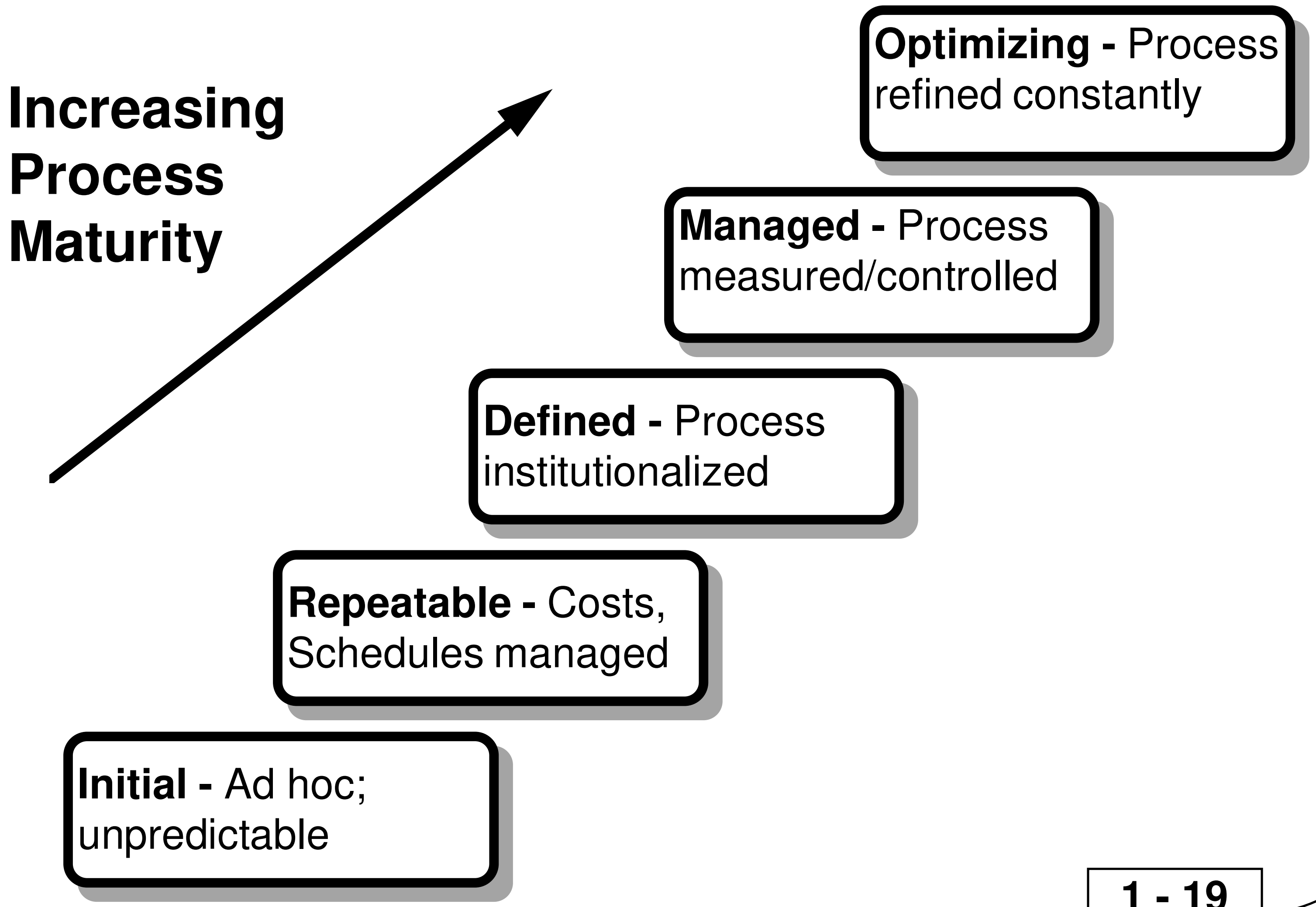
---

3. MAINTENANCE PHASE
   - Correction
   - Adaptation
   - Enhancement

Software Engineering Capability and Its Measurement

- **The maturity of an organization's software engineering capability can be measured in terms of the degree to which the outcome of the process by which software is developed can be predicted.**
  - ❍ **Predict the amount of time required to develop a software artifact**
  - ❍ **Predict the resources (number of people, amount of disk space, *etc.*) required to develop a software artifact**
  - ❍ **Predict the cost of developing a software artifact**
- **The *process* and the *technology* go hand in hand.**
- **One method of measurement is the *Capability Maturity Model for Software* developed by the Software Engineering Institute.**

Software Engineering Capability and Its Measurement


Increasing Process Maturity
Optimizing - Process refined constantly
Managed - Process measured/controlled
Defined - Process institutionalized
Repeatable - Costs, Schedules managed
Initial - Ad hoc; unpredictable

SOFTWARE COMPLEXITY, OBJECT-ORIENTED REQUIREMENTS ANALYSIS (OORA), AND OBJECT-ORIENTED DESIGN (OOD)

✓ The Inherent Complexity of Software

✓ The Attributes of Complex Systems

✓ Canonical Form of a Complex System

✓ On Designing Complex Systems

The Inherent Complexity of Software

A *simple* software system is:

- completely specified or nearly so with a small set of behaviors
- completely understandable by a single person
- one that we can afford to throw away and replace with entirely new software when it comes time to repair them or extend their functionality

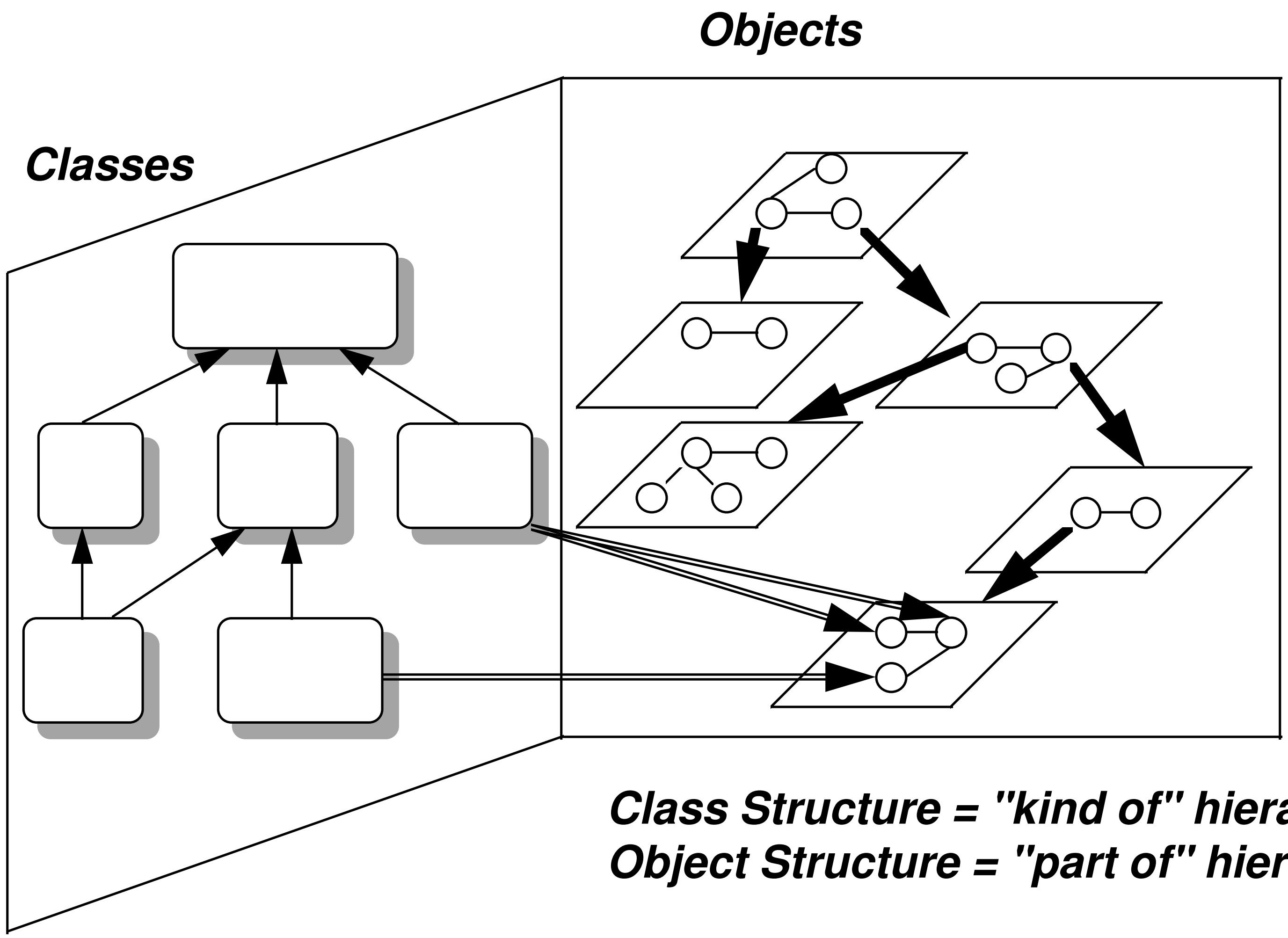
A *complex* software system (*industrial-strength software*) is:

- one which exhibits a rich set of behaviors
- extremely difficult, if not impossible, for an individual to comprehend all of its aspects - exceeds the average human intellectual capacity
- one that we can NOT afford to throw away and replace with entirely new software, so we patch it, maintain out-of-date development environments for it, and carefully control changes to it and its operational environment

The Attributes of Complex Systems

1. A complex system is implemented in a hierarchical structure.
2. The determination of this hierarchy, selecting upper-level subsystems, lower-level subsystems, and primitive components, is relatively arbitrary, largely up to the discretion of the designer of the system.
3. Linkages within the components of a system are usually stronger than linkages between the components of a system.
4. Complex systems are often composed of only a few different classes of subsystems, although there may be many instances of each class.
5. Working complex systems have invariably evolved from working simpler systems. A complex system designed from scratch has never worked and cannot be patched to make it work.

Canonical Form of a Complex System



Class Structure = "kind of" hierarchy
Object Structure = "part of" hierarchy

On Designing Complex Systems

Requirements Analysis - the disciplined approach used to understand a problem

Design - the disciplined approach used to devise a solution to a problem

The Purpose of Design

To construct a system that:

- satisfies a given specification
- conforms to limitations of the target
- meets constraints on performance and resource usage
- satisfies a given set of design criteria on the artifact
- satisfies restrictions on the design process itself, such as cost and schedule

Elements of Design

Notation - the language of expression

Process - the steps taken for the orderly construction of the design

Tools - the artifacts that support the design process by reducing the level of effort