SOFTWARE and SOFTWARE ENGINEERING

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

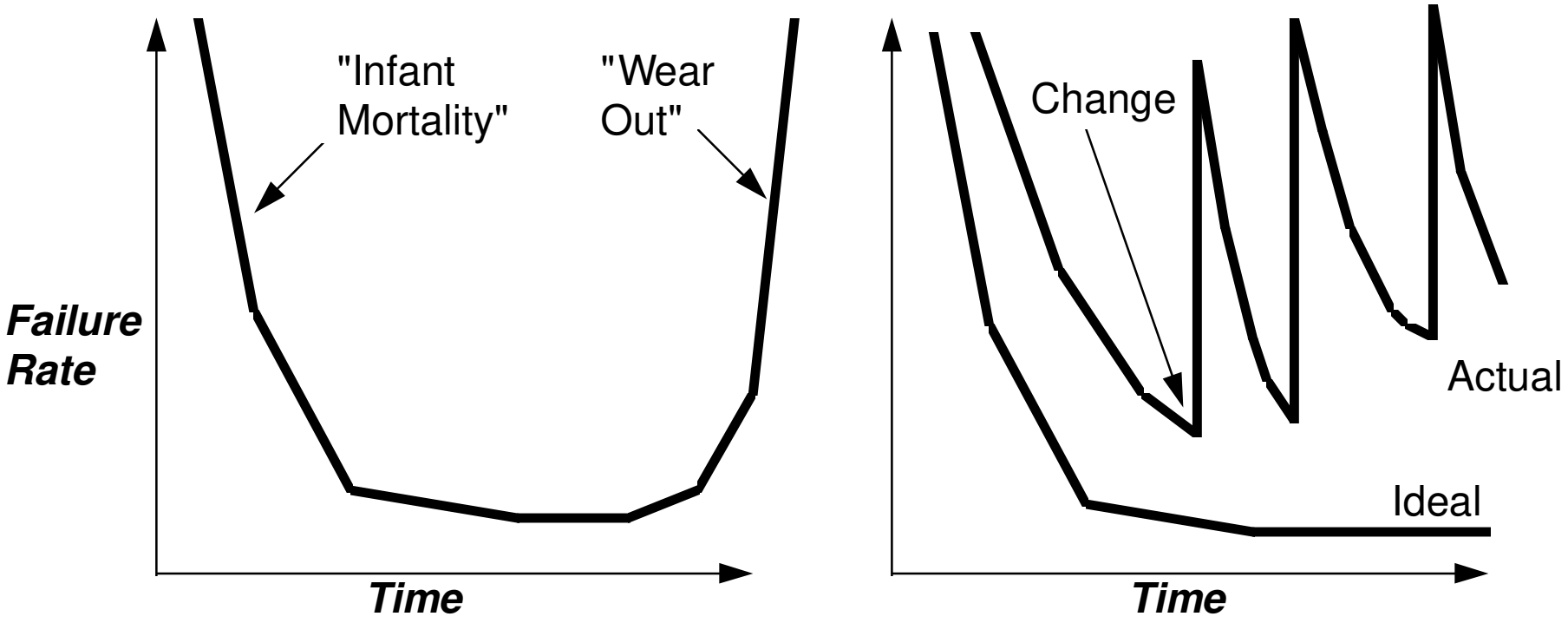
THE NATURE OF SOFTWARE

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

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



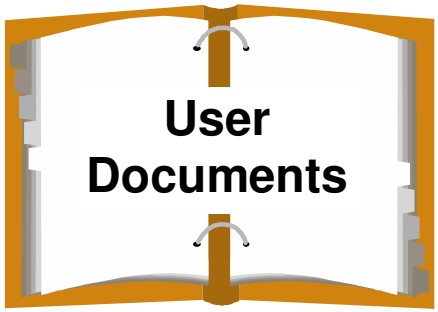
FAILURE CURVE FOR HARDWARE

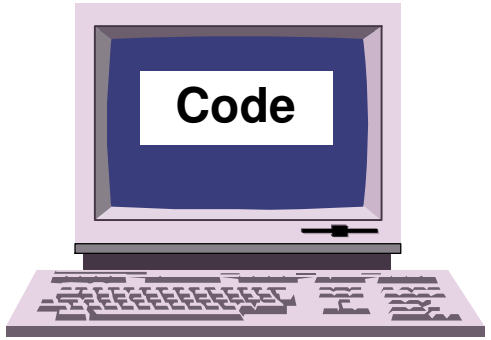
FAILURE CURVE FOR SOFTWARE

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 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

Software Application Domains

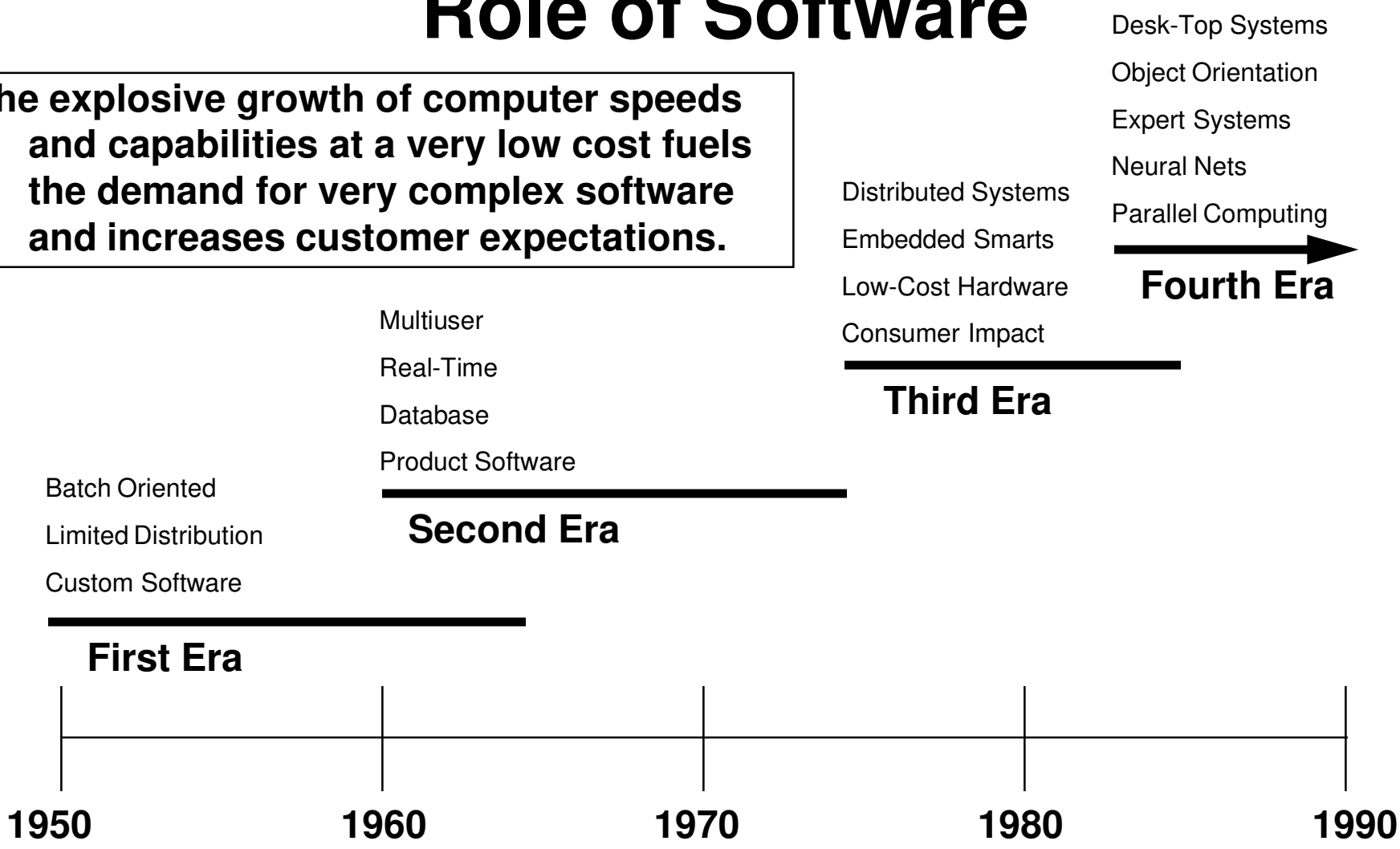
- System
  - compilers
  - editors
  - file management
- Real-time
  - machine control
  - auto controls
  - jet engine control
- Business
  - databases
  - stock management
  - point-of-sale terminals
- Embedded
  - appliance control
  - FPGA programs
  - auto controls
- Engineering and Scientific
  - simulation
  - computer-aided design
  - "number crunching"
- Personal Computer
  - all non-realtime above
- Artificial Intelligence
  - expert systems
  - neural networks

HISTORY OF SOFTWARE DEVELOPMENT

- ✓ **Role of Software**
- ✓ **Industrial View**

Role of Software

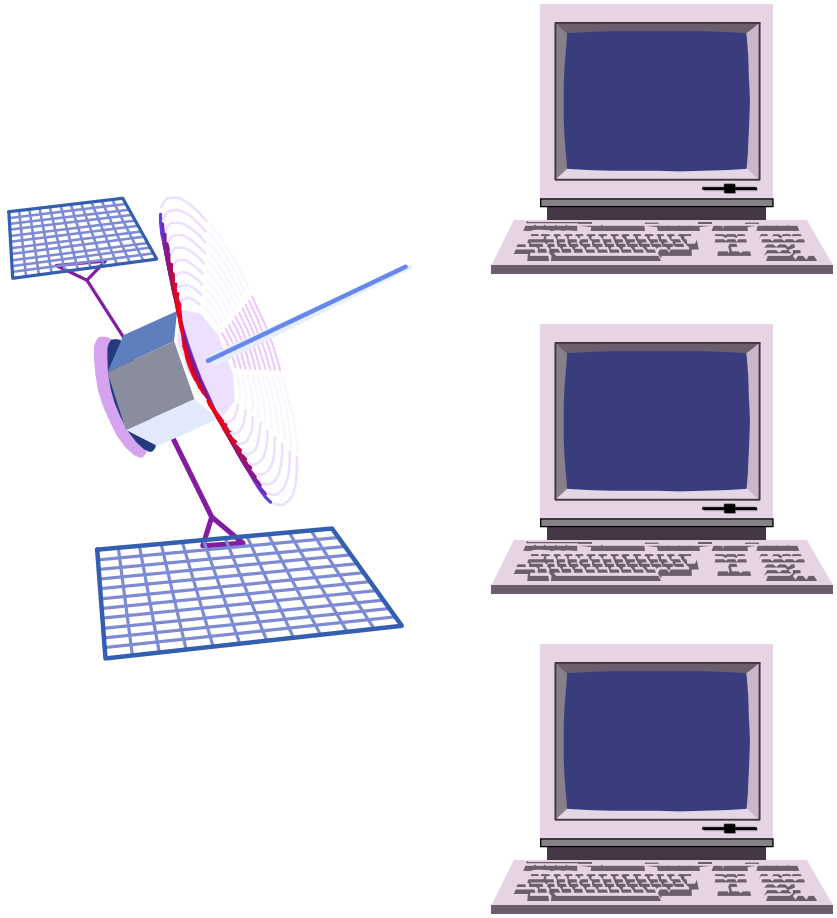
The explosive growth of computer speeds and capabilities at a very low cost fuels the demand for very complex software and increases customer expectations.



Role of Software, Continued

Where Do We Go From Here?

- **Parallel computing to extend speed of computation**
- **Object-oriented methods of software design**
- **Software frameworks evolve to handle larger and multiprogram systems**
- **Heavy dependence on graphics interfaces**
- **Artificial intelligence and neural computing become useful**
- **National computing motivates huge software systems**
- **Advanced programming languages**



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?

PROBLEMS WITH SOFTWARE DEVELOPMENT

- ✓ **Problems**
- ✓ **Causes**

Problems

1. We have little data on the software development process.
2. Customers are often dissatisfied with the software they get.
3. Software quality is hard to define and measure.
4. Existing software is often very difficult to maintain.

Can these problems be overcome?

Causes

- No spare parts to replace, so an error in the original software is also in every copy.
- Software quality is a human problem.
- Project managers often have no software development experience.
- Software developers often have little or no formal training in engineering the development of the software product.
- Resistance to change from programming as an art to programming as an engineering task can be significant.

SOFTWARE MYTHS

- ✓ Customer Myths
- ✓ Developer Myths
- ✓ Management Myths

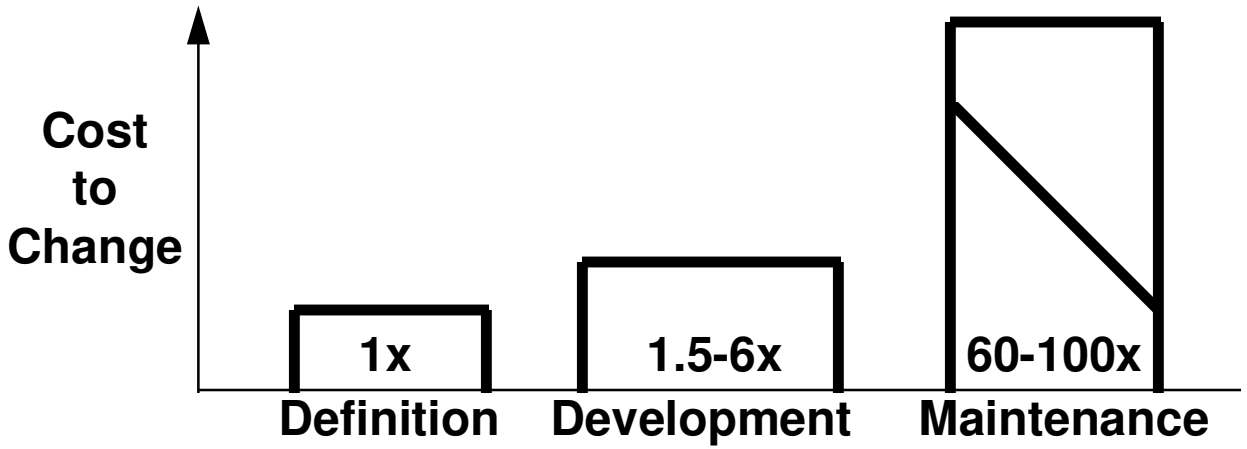
Customer Myths

Myth

- **A general statement of objectives is enough to get going. Fill in the details later.**
- **Project requirements continually change, but change can be easily accommodated because software is flexible.**

Reality

- **Poor up-front definition of the requirements is *THE* major cause of poor and late software.**
- **Cost of the change to software in order to fix an error increases dramatically in later phases of the life of the software.**



Developer Myths

Myth

- Once a program is written and works, the developer's job is done.
- Until a program is running, there is no way to assess its quality.
- The only deliverable for a successful project is a working program.

Reality

- 50%-70% of the effort expended on a program occurs after it is delivered to the customer.
- Software reviews can be more effective in finding errors than testing for certain classes of errors.
- A software configuration includes documentation, regeneration files, test input data, and test results data.

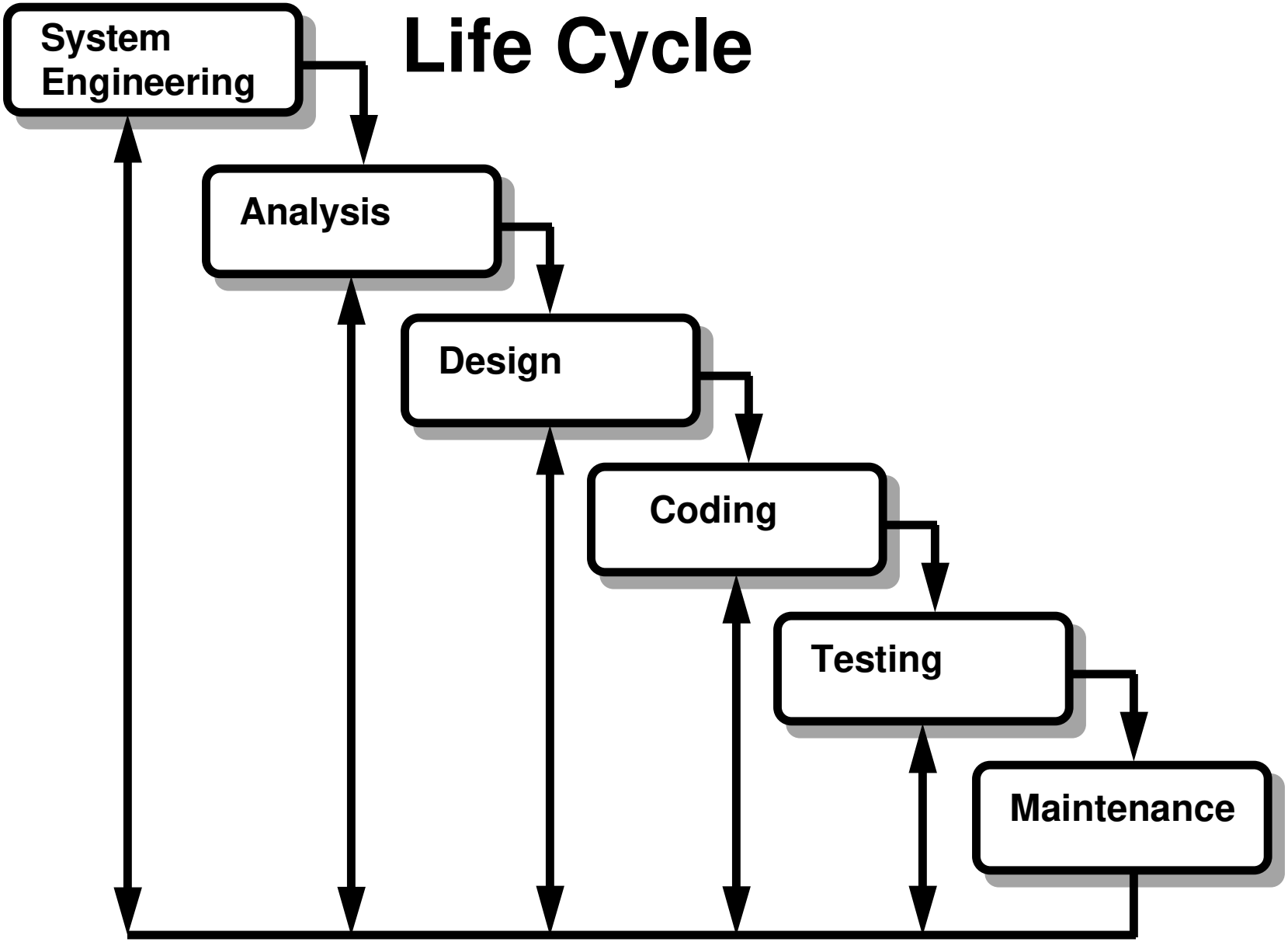
Management Myths

Myth	*Reality*
● Books of standards exist in-house so software will be developed satisfactorily.	● Books may exist, but they are usually not up to date and not used.
● Computers and software tools that are available in-house are sufficient.	● CASE tools are needed but are not usually obtained or used.
● We can always add more programmers if the project gets behind.	● "Adding people to a late software project makes it later." -- *Brooks*

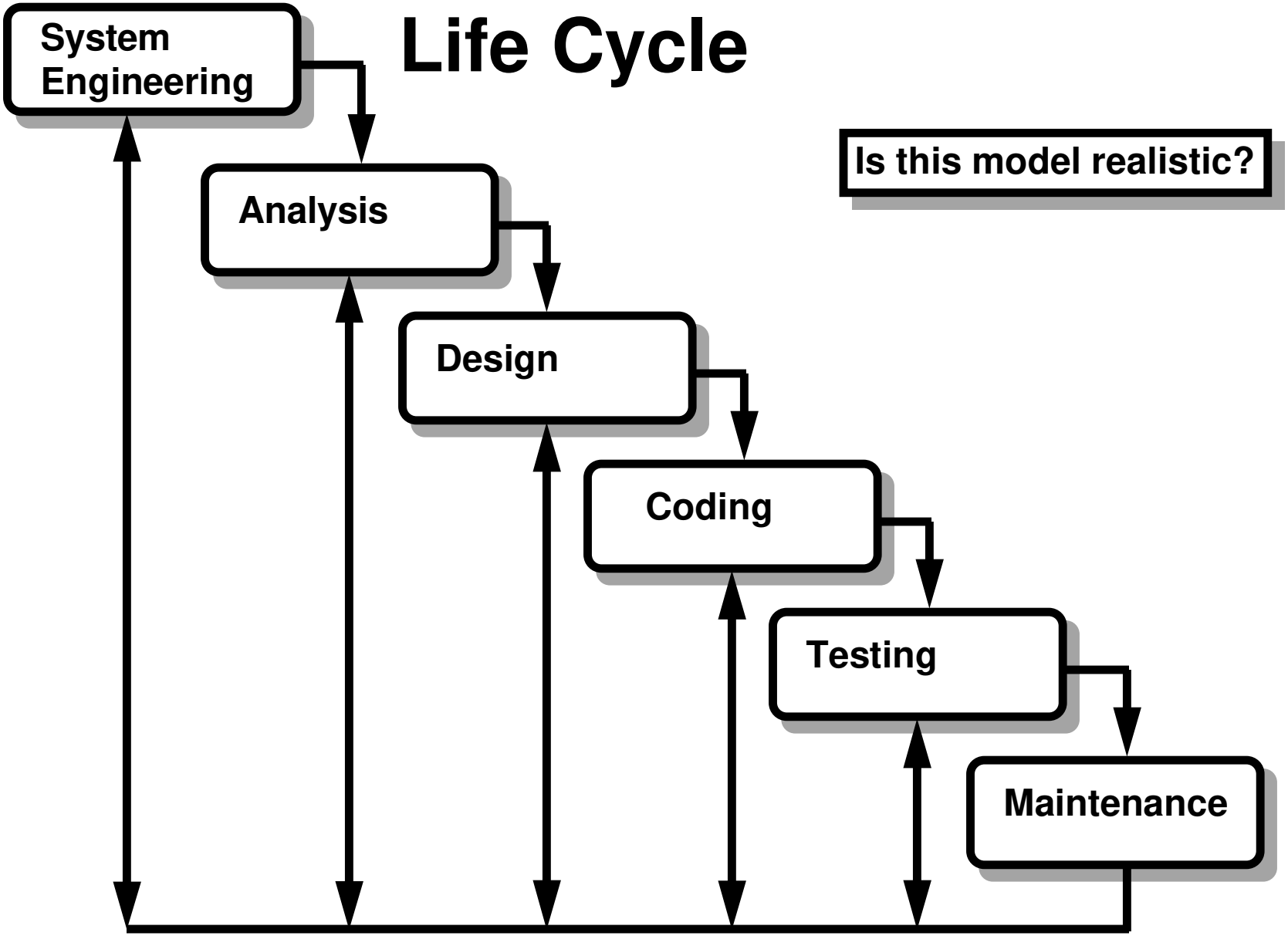
SOFTWARE ENGINEERING PARADIGMS

- ✓ Life Cycle
- ✓ Prototyping Model
- ✓ Spiral Model
- ✓ Fourth Generation Techniques
- ✓ Combining Paradigms
- ✓ Generic Paradigm

Life Cycle

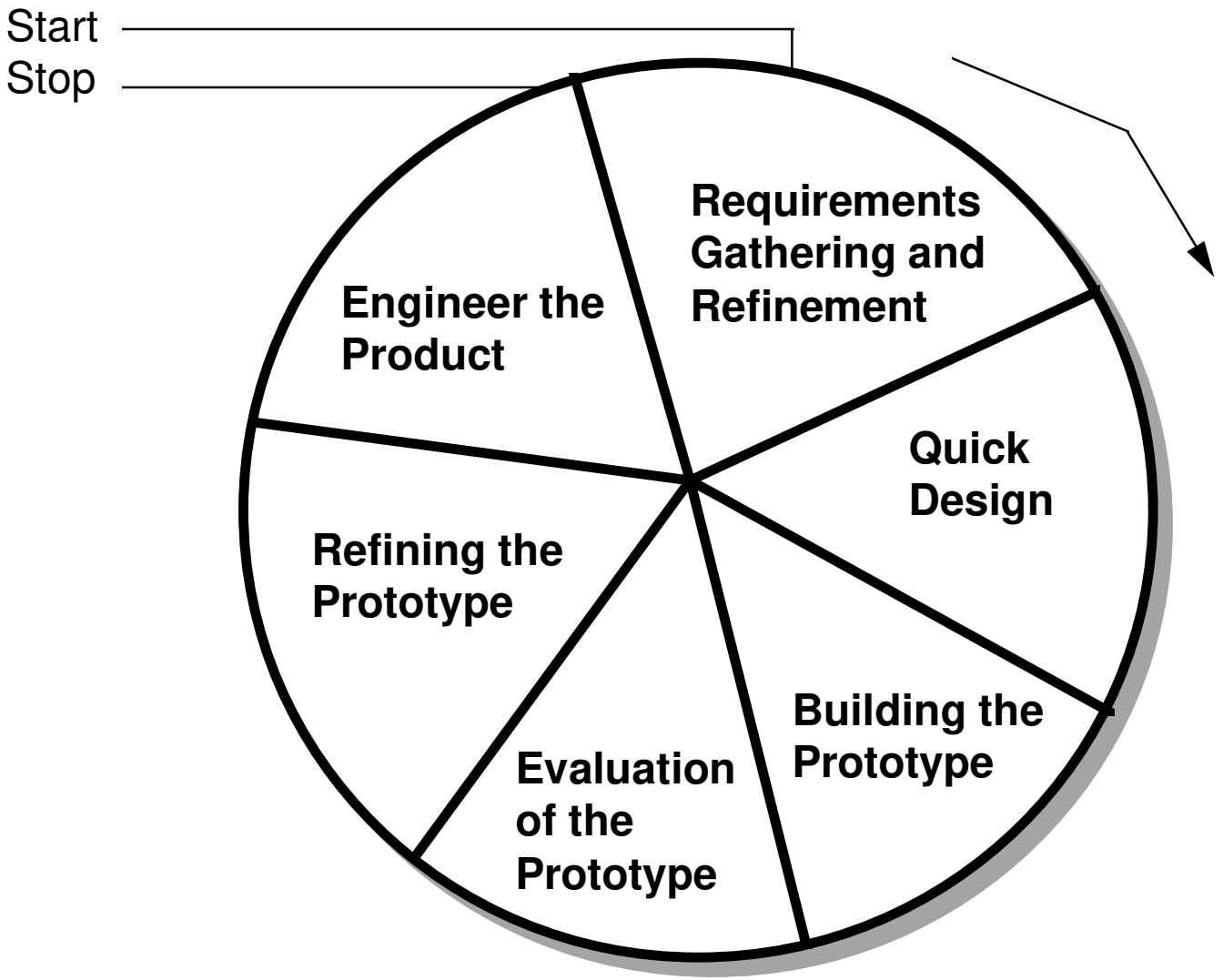

System Engineering
Analysis
Design
Coding
Testing
Maintenance

Life Cycle

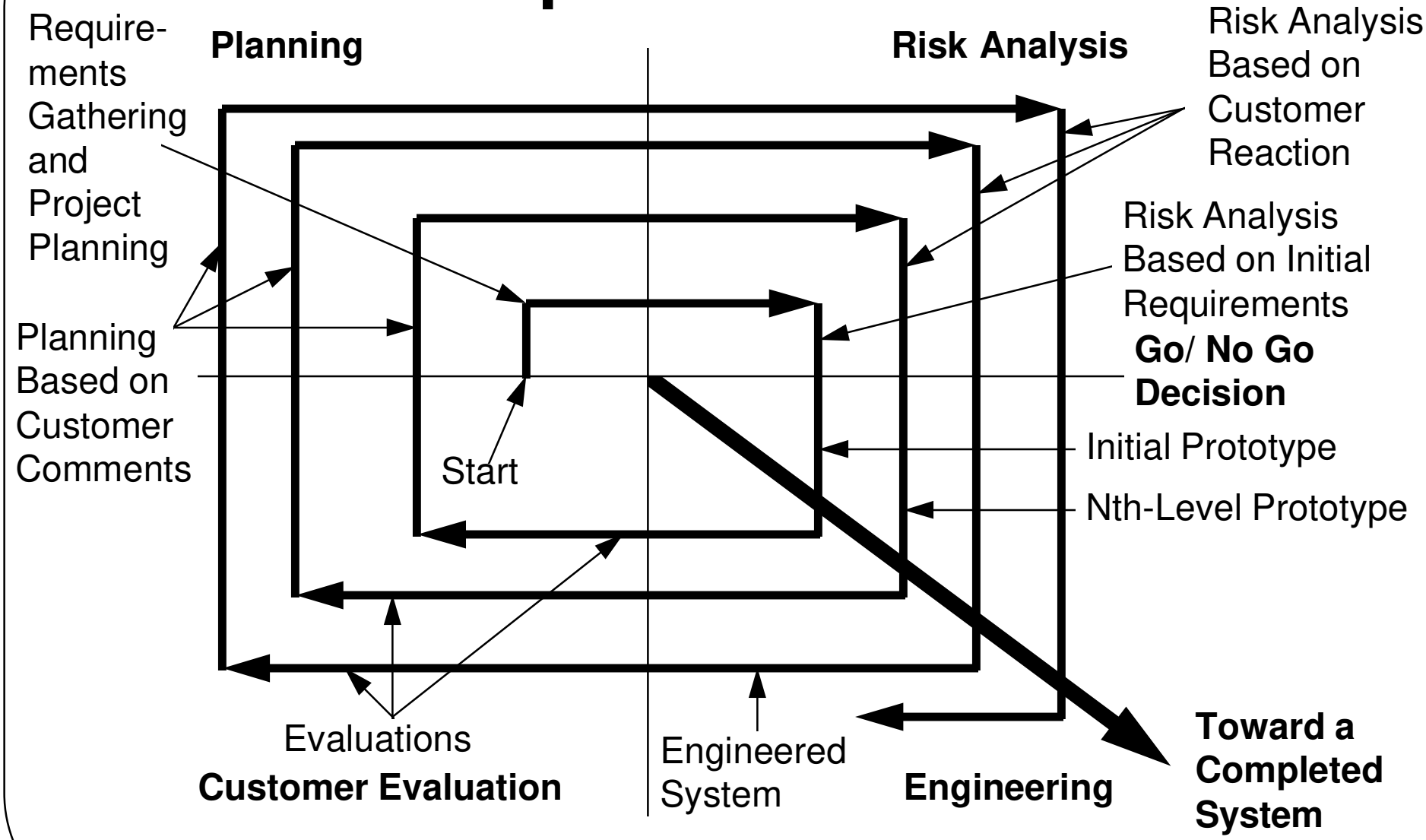


Is this model realistic?

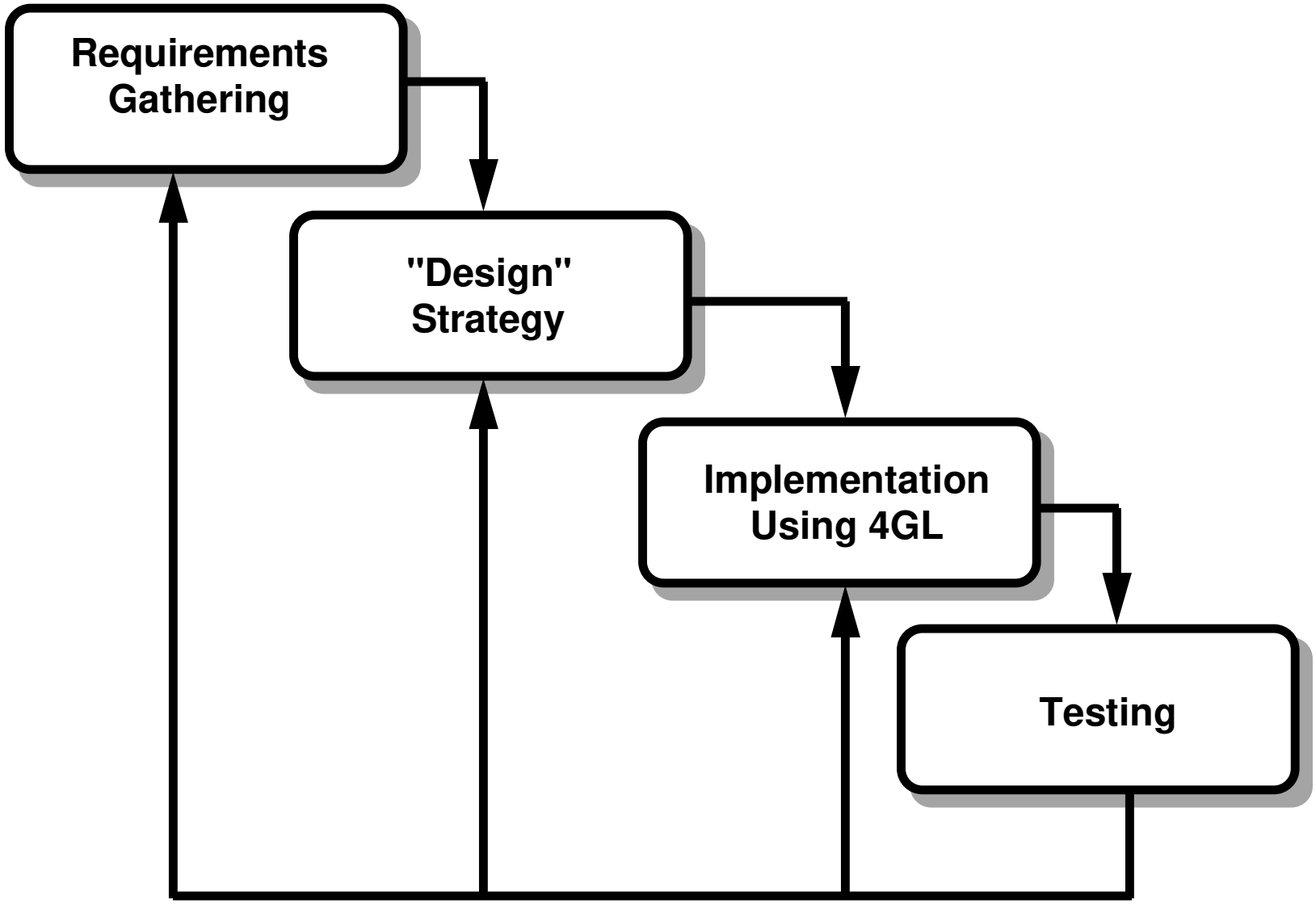
Prototyping Model



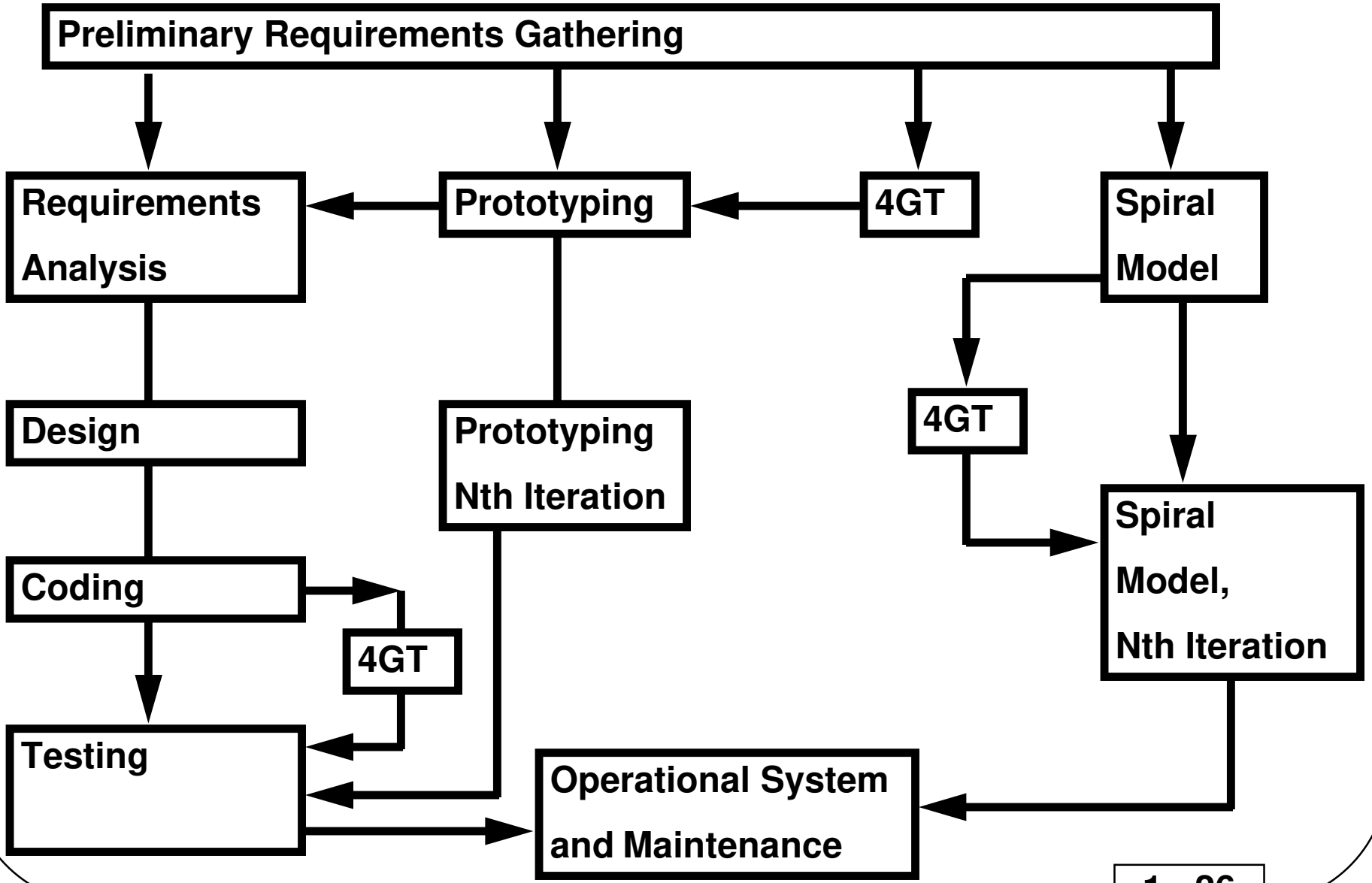
Spiral Model



Fourth Generation Techniques



Combining Paradigms



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 TECHNOLOGY

- ✓ What is Software Engineering?
- ✓ Software Engineering Capability and Its Measurement
- ✓ Ada Technology

What Is Software Engineering?

Methods

- Analysis
- Design
- Coding
- Testing
- Maintenance

Procedures

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

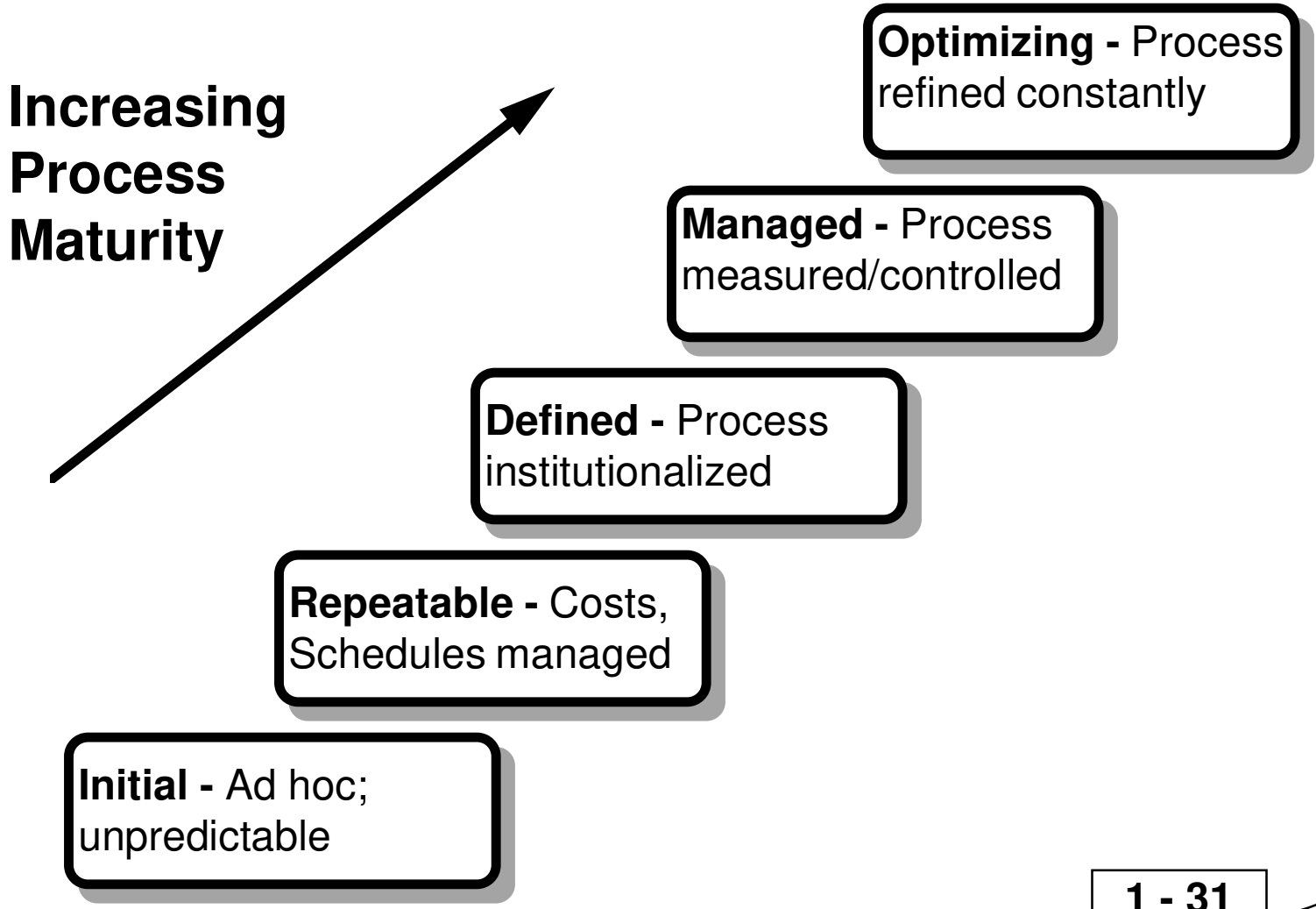
Computer-Aided Software Engineering **(CASE)**

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

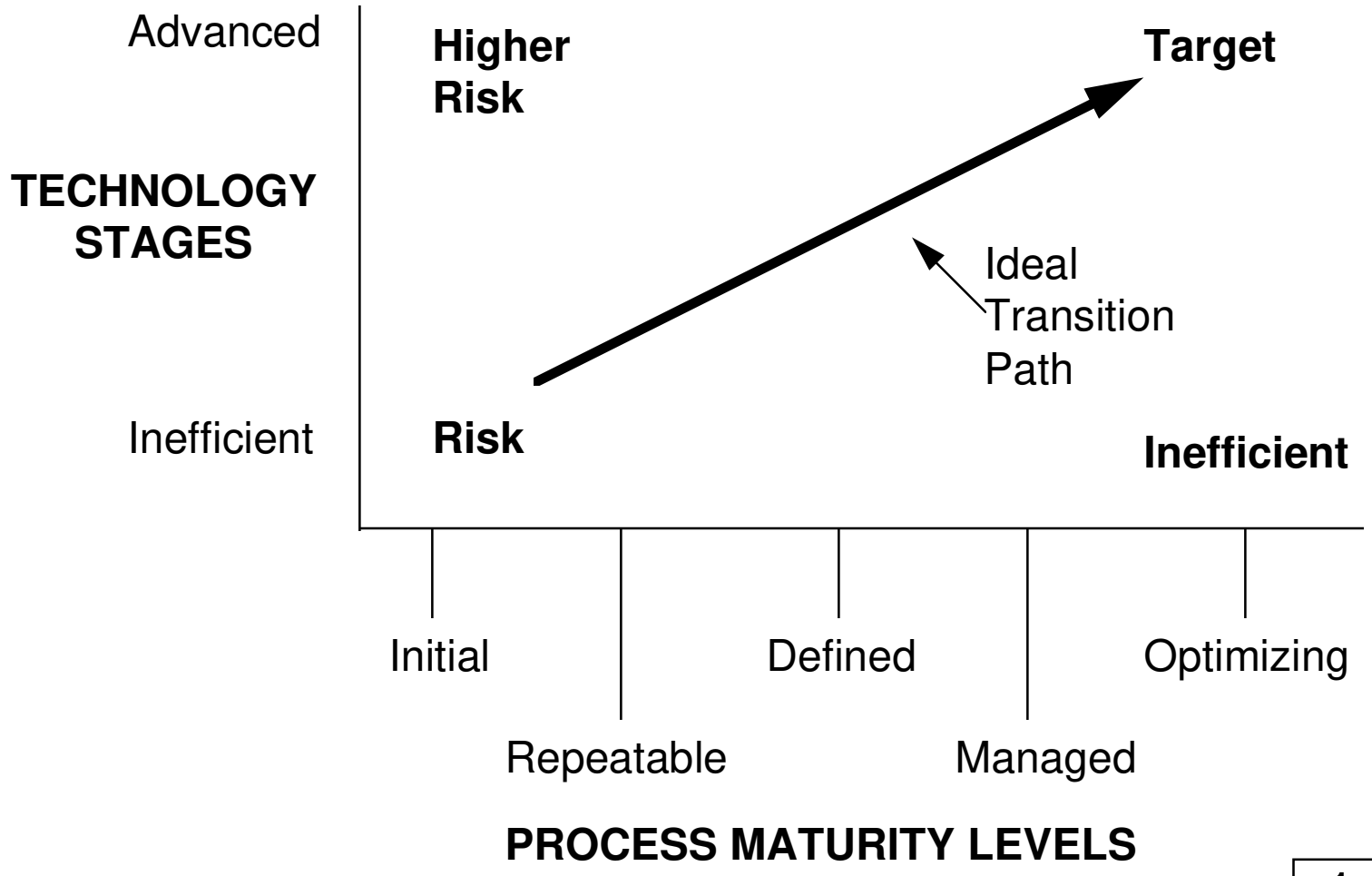
Software Engineering Capability and Its Measurement

- 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.

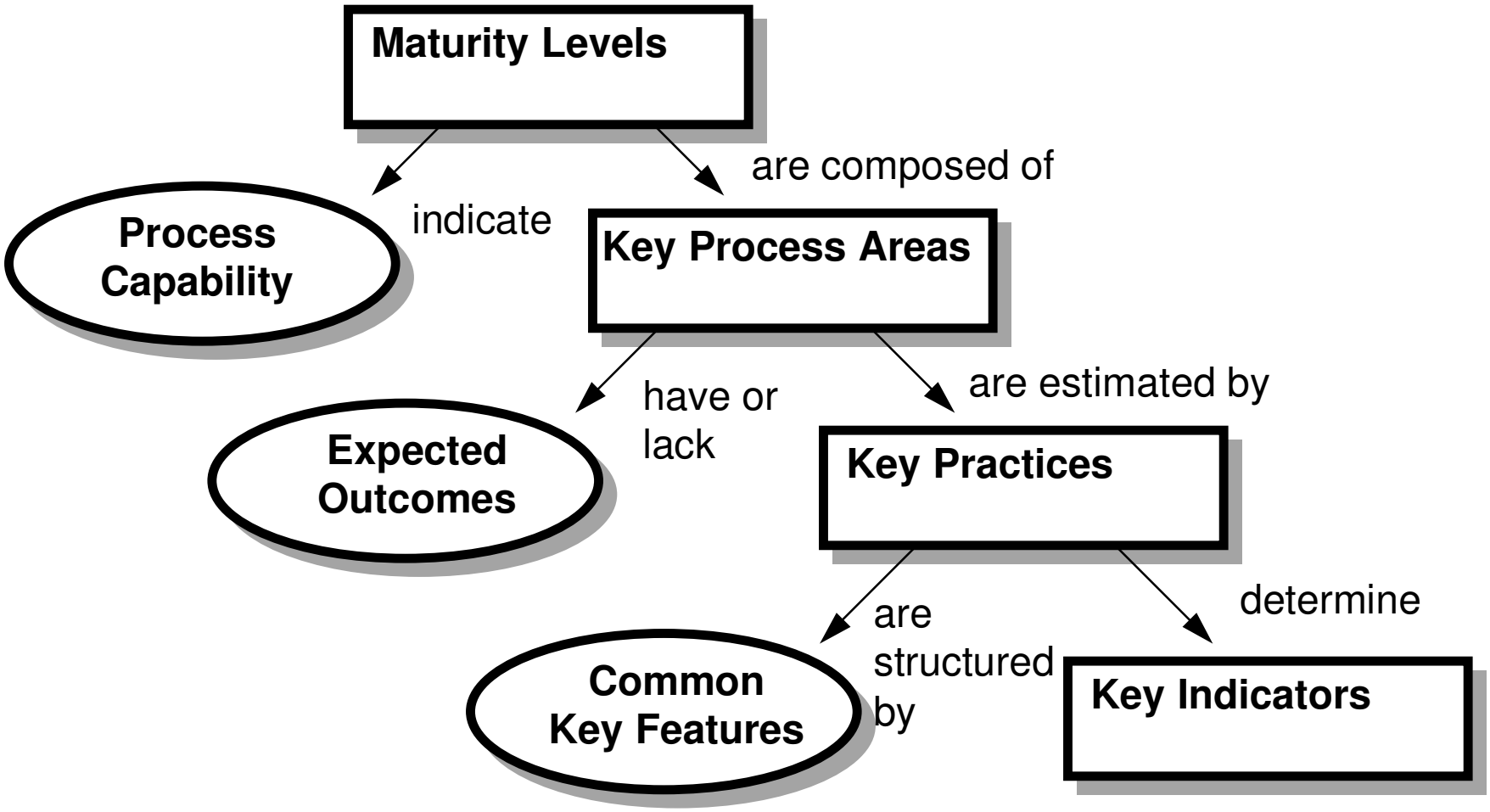
Software Engineering Capability and Its Measurement



Software Engineering Capability and Its Measurement



Software Engineering Capability and Its Measurement



Key Process Areas by Level
Level 2 (Repeatable)

- **Requirements Management**
- **Software Project Planning**
- Software Project Tracking and Oversight
- Software Subcontract Management
- Software Quality Assurance
- Software Configuration Management

Key Process Areas by Level
Level 2 (Repeatable), Continued

- Requirements Management
- Software Project Planning
- **Software Project Tracking and Oversight**
- **Software Subcontract Management**
- Software Quality Assurance
- Software Configuration Management

Key Process Areas by Level
Level 2 (Repeatable), Continued

- Requirements Management
- Software Project Planning
- Software Project Tracking and Oversight
- Software Subcontract Management
- **Software Quality Assurance**
- **Software Configuration Management**

Key Process Areas by Level
Level 3 (Defined)

- **Organization Process Focus**
- **Organization Process Definition**
- Training Program
- Integrated Software Management
- Software Product Engineering
- Intergroup Coordination
- Peer Reviews

Key Process Areas by Level
Level 3 (Defined), Continued

- Organization Process Focus
- Organization Process Definition
- **Training Program**
- **Integrated Software Management**
- Software Product Engineering
- Intergroup Coordination
- Peer Reviews

Key Process Areas by Level
Level 3 (Defined), Continued

- Organization Process Focus
- Organization Process Definition
- Training Program
- Integrated Software Management
- **Software Product Engineering**
- **Intergroup Coordination**
- Peer Reviews

Key Process Areas by Level
Level 3 (Defined), Continued

- Organization Process Focus
- Organization Process Definition
- Training Program
- Integrated Software Management
- Software Product Engineering
- Intergroup Coordination
- **Peer Reviews**

Key Process Areas by Level
Level 4 (Managed)

- **Process Measurement and Analysis**
- **Quality Management**

Key Process Areas by Level
Level 5 (Optimizing)

- Defect Prevention
- Technology Innovation
- Process Change Management

Ada Technology

- ***Ada*** **is a computer programming language specifically designed to support software engineering.**
- **Some of Ada's features include:**
  - **All of the normal constructs for looping, branching, flow control, and subprogram construction**
  - **Support for enumeration types, integers, floating point, fixed point, characters, strings, arrays, records, and user-defined data types**
  - **Support for algorithm templates (called generics) which allow algorithms to be expressed without concern for the kind of data on which the algorithm is applied**
  - **Support for interrupts and concurrent processing**
  - **Support for low-level control, such as memory allocation**
- **Ada is a *design* language as well as a *programming* language.**
- **Ada is designed to be read by Ada programmers and non-programmers.**

Ada Technology


```
with System;
package Sensor is
   type Device is private;
   -- Abstract concept of a sensor
   procedure Define (S : in out Device;
      Where : in System.Address);
   -- Associate a sensor with memory
   function Read(S : in Device)
         return Integer;
   -- Return sensed value
private
   -- details omitted
end Sensor;
```

Ada Technology

- From the software engineering perspective, Ada helps by acting as something much more than a programming language; Ada can be used as a common language for communicating:
  - Some aspects of the requirements
  - Some aspects of the design
  - All aspects of the code
- In particular, by using Ada as a *design language*, code is simply realized as a complete, detailed elaboration of a design.
- For large, multi-person teams, Ada can be used as an exact, precise way to communicate requirements and design information -- often in a form which may be syntactically checked by a compiler. Ada is much better than conventional English in this regard.

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
- ✓ Bringing Order to Chaos
- ✓ 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**

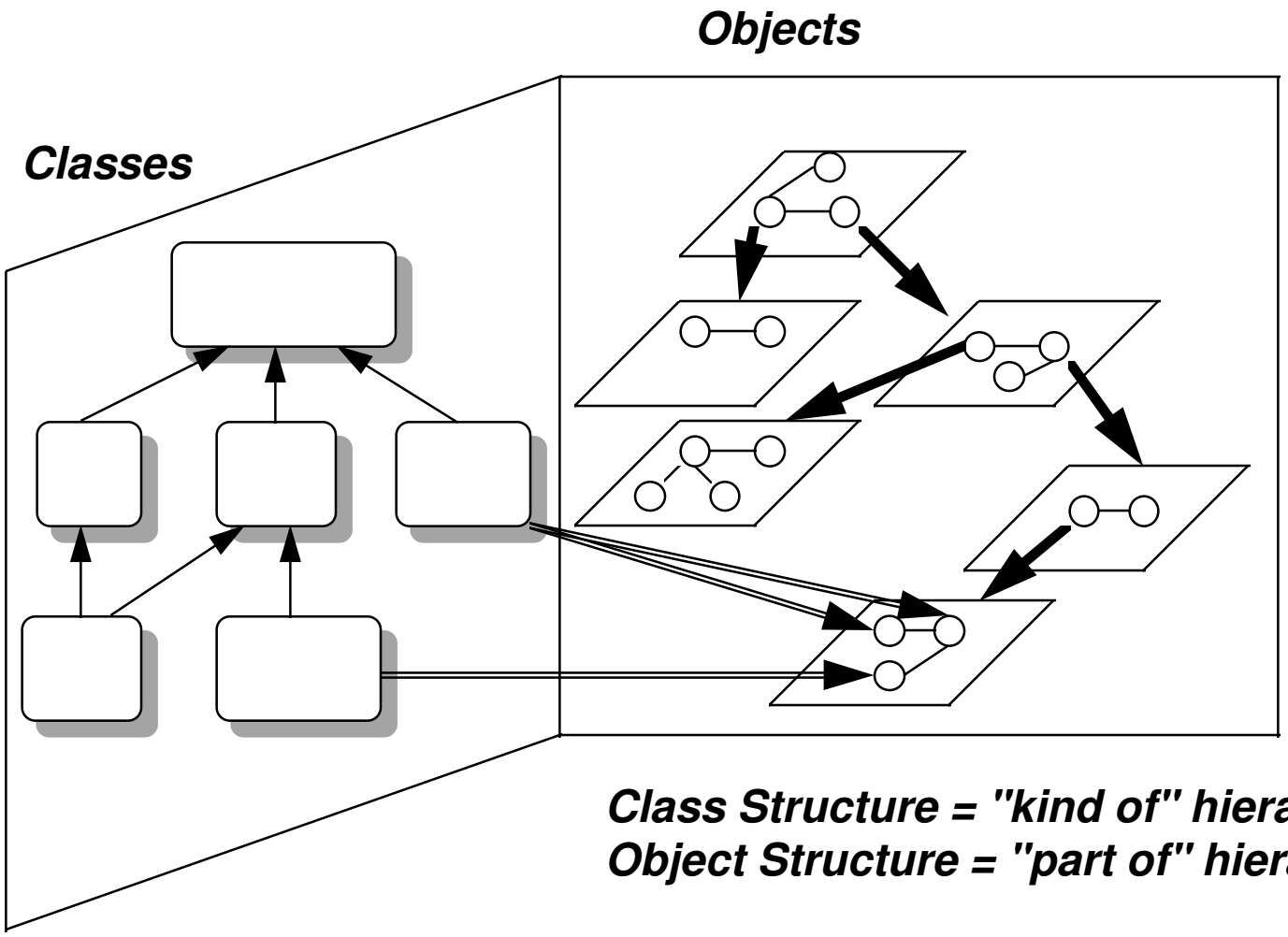
A *complex* software system (*industrial-strength software*) is:

- **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

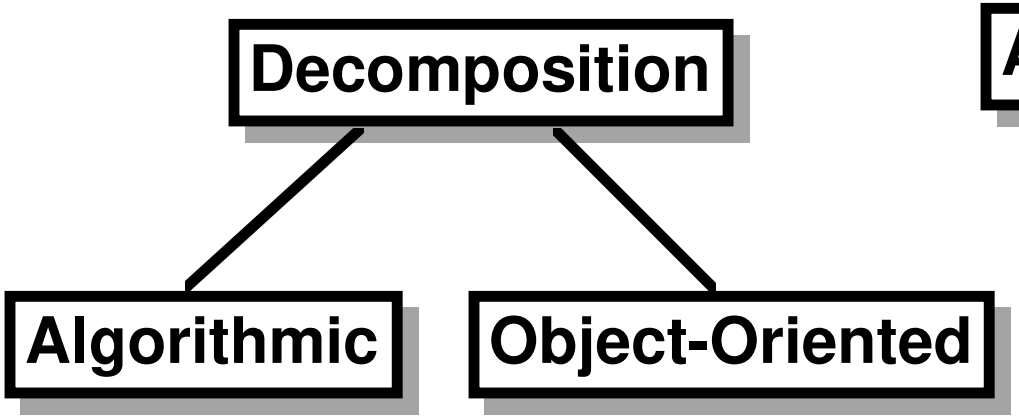
1. A complex system is implemented in a hierarchical structure.
2. The determination of this hierarchy, selecting upper-level subsystems, lower-level subsystems, and primitive components, is relatively arbitrary, largely up to the discretion of the designer of the system.
3. Linkages within the components 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

Bringing Order to Chaos



Abstraction and Hierarchy

The Role of Abstraction

- Reducing the number of concepts to comprehend concurrently
- 7 +/- 2

The Role of Hierarchy

- Understanding the relationships between entities (object structure)
- Understanding the redundancy in the entities (class structure)

The Role of Decomposition

- "Divide and Conquer"
- Break a large system into a number of smaller subsystems and so on until the subsystems are manageable and understandable
- Two views: algorithmic and object-oriented

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**