

SOFTWARE and SOFTWARE ENGINEERING

 **Engineered Software**

 **Controlled Configuration Item**

 **History of Software
Development**

 **Software as a Business
Opportunity**

 **Problems in Software
Development**

 **Myths about Software
Development**

 **Approaches to Software
Development**

 **Engineered Software
Development Models**

 **Software Engineering
Technology**






TOPICS

**The Nature and History of Software
Development**

Problems with Software Development

**Software Engineering Paradigms and
Technology**

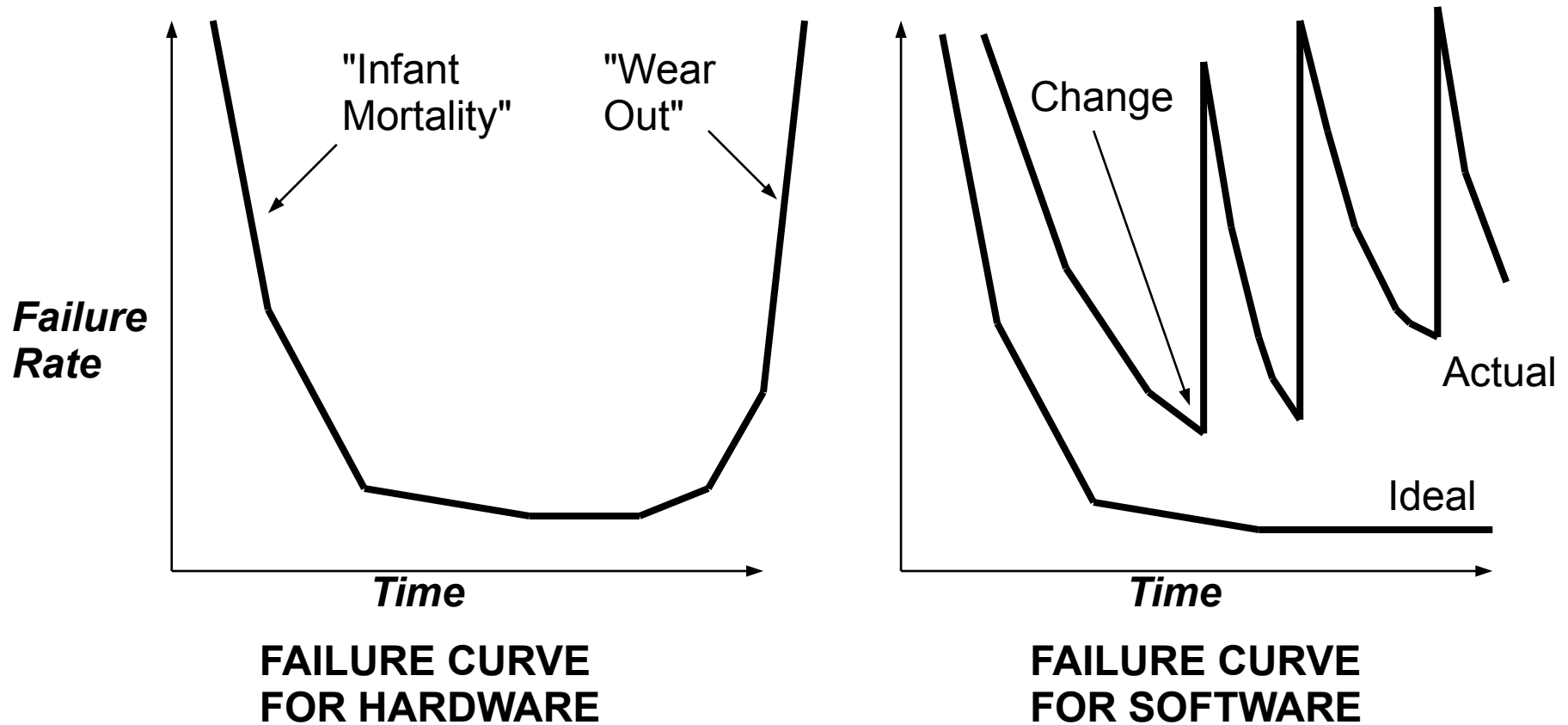
THE NATURE OF SOFTWARE

-  **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



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 Development Activities

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

Operating Systems

● Real Time

machine control

auto controls

Business

databases

stock management

● Personal Computer

all non-realtime above

Embedded

appliance control

FPGA programs

auto controls

Engineering and Scientific

simulation

computer-aided design

"number crunching"

Artificial Intelligence

expert systems

neural networks

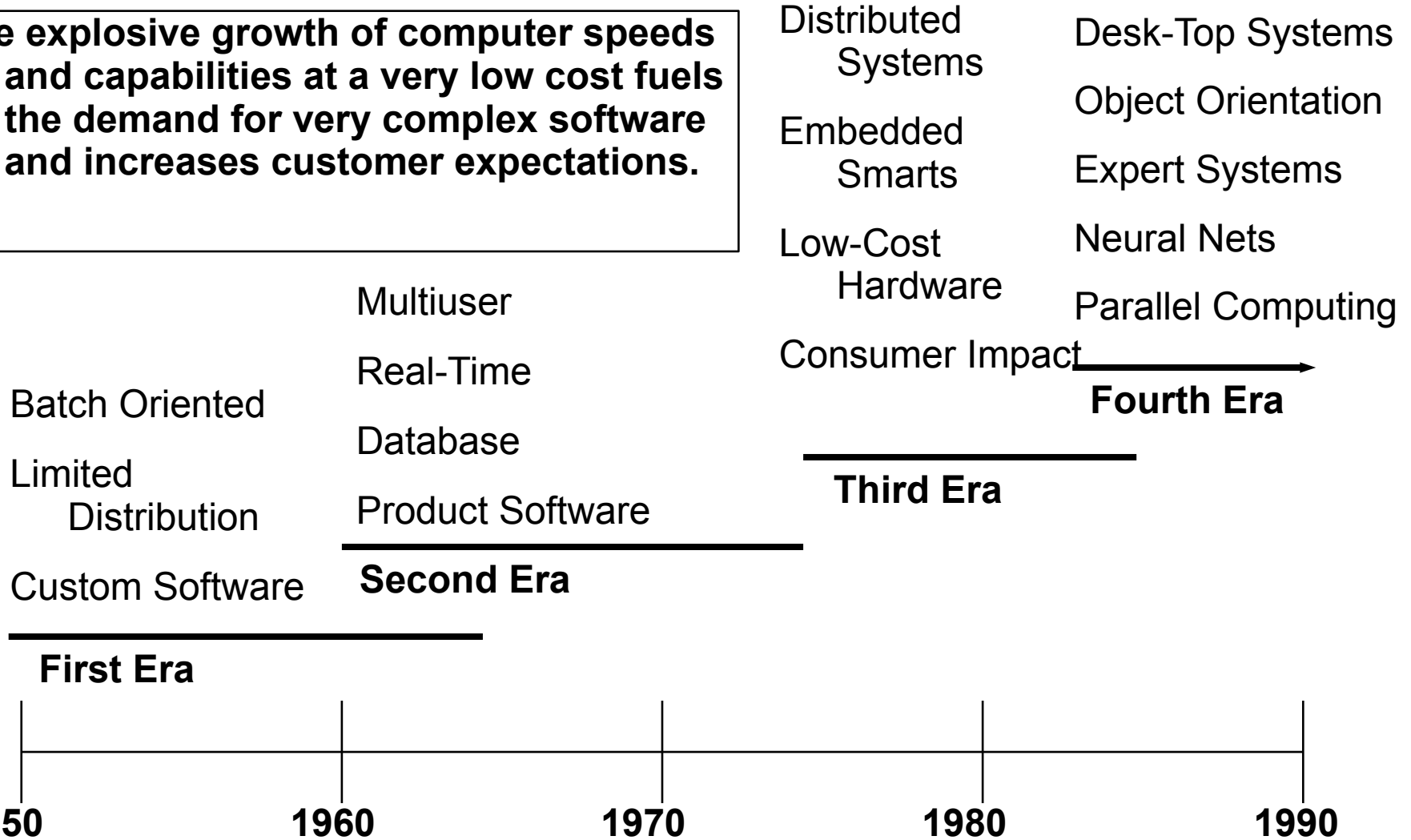
HISTORY OF SOFTWARE DEVELOPMENT

 **Role of Software**

 **Industrial View**

Role of Software

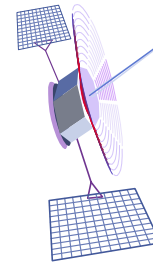
The explosive growth of computer speeds and capabilities at a very low cost fuels the demand for very complex software and increases customer expectations.



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

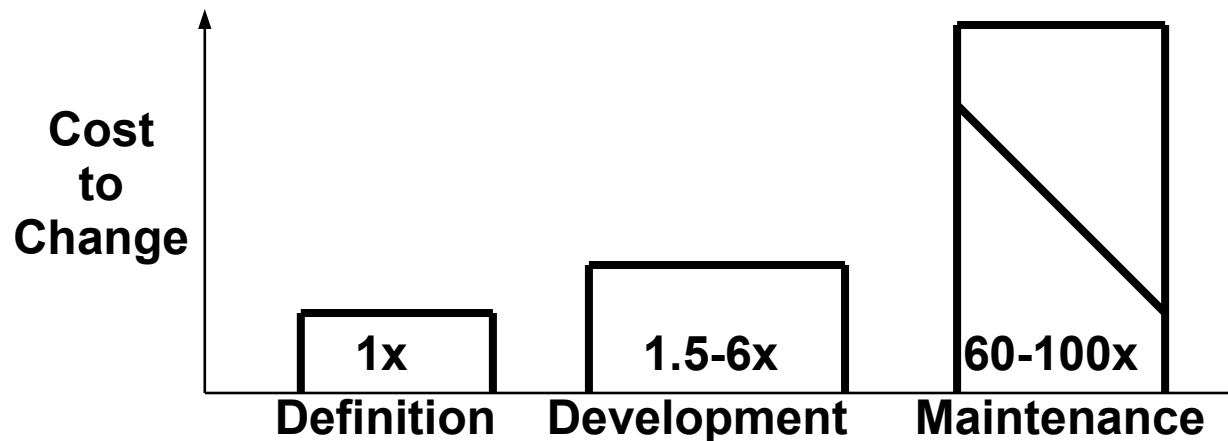
Reality

❑ A general statement of objectives is enough to get going. Fill in the details later.

❑ Poor up-front definition of the requirements is *THE* major cause of poor and late software.

❑ Project requirements continually change, but change can be easily accommodated because software is flexible.

❑ 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

- ❑ Books of standards exist in-house so software will be developed satisfactorily.
- ❑ Computers and software tools that are available in-house are sufficient.
- ❑ We can always add more programmers if the project gets behind.

Reality

- ❑ Books may exist, but they are usually not up to date and not used.
- ❑ CASE tools are needed but are not usually obtained or used.
- ❑ "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

**System
Engineering**

Analysis

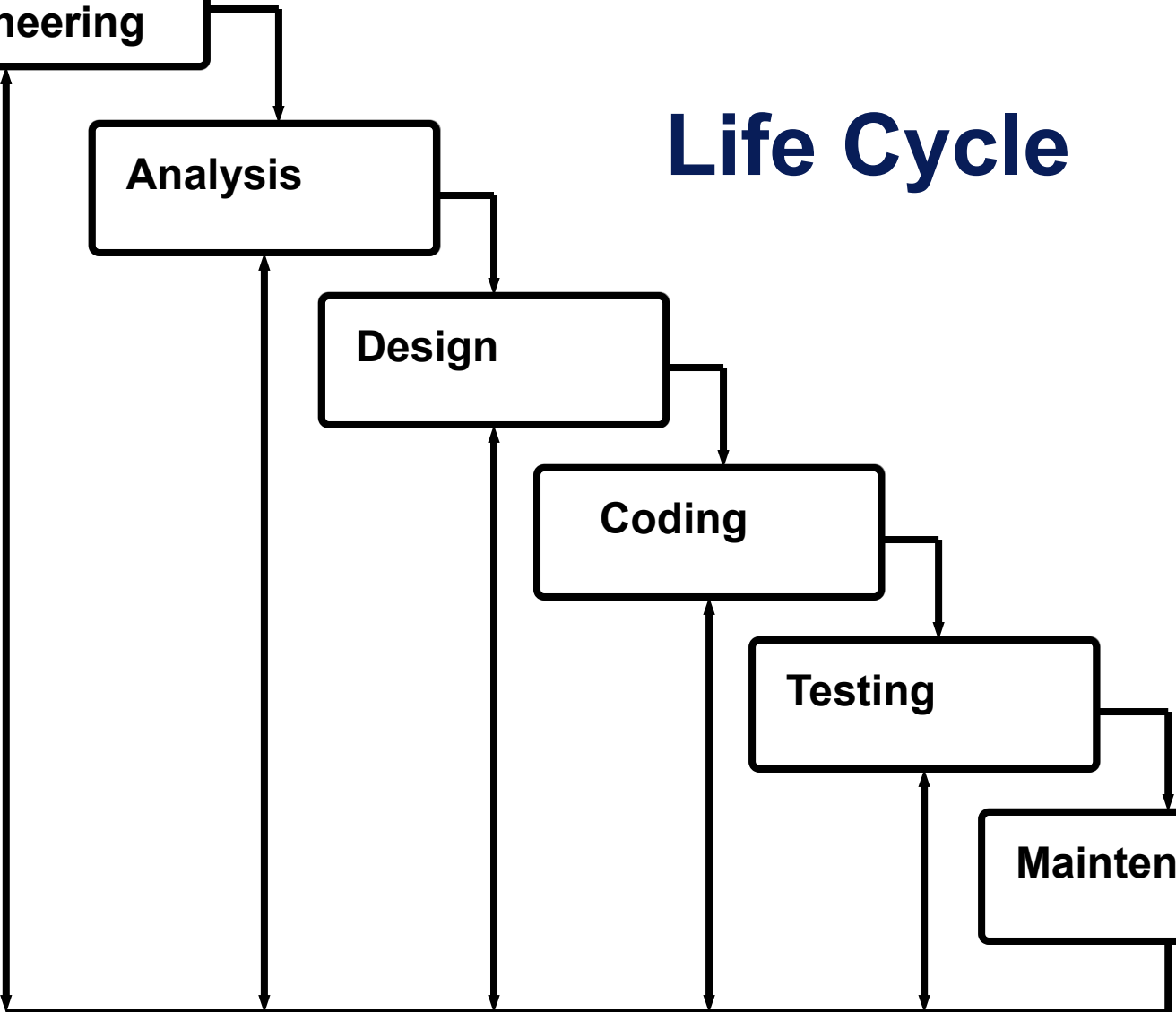
Design

Coding

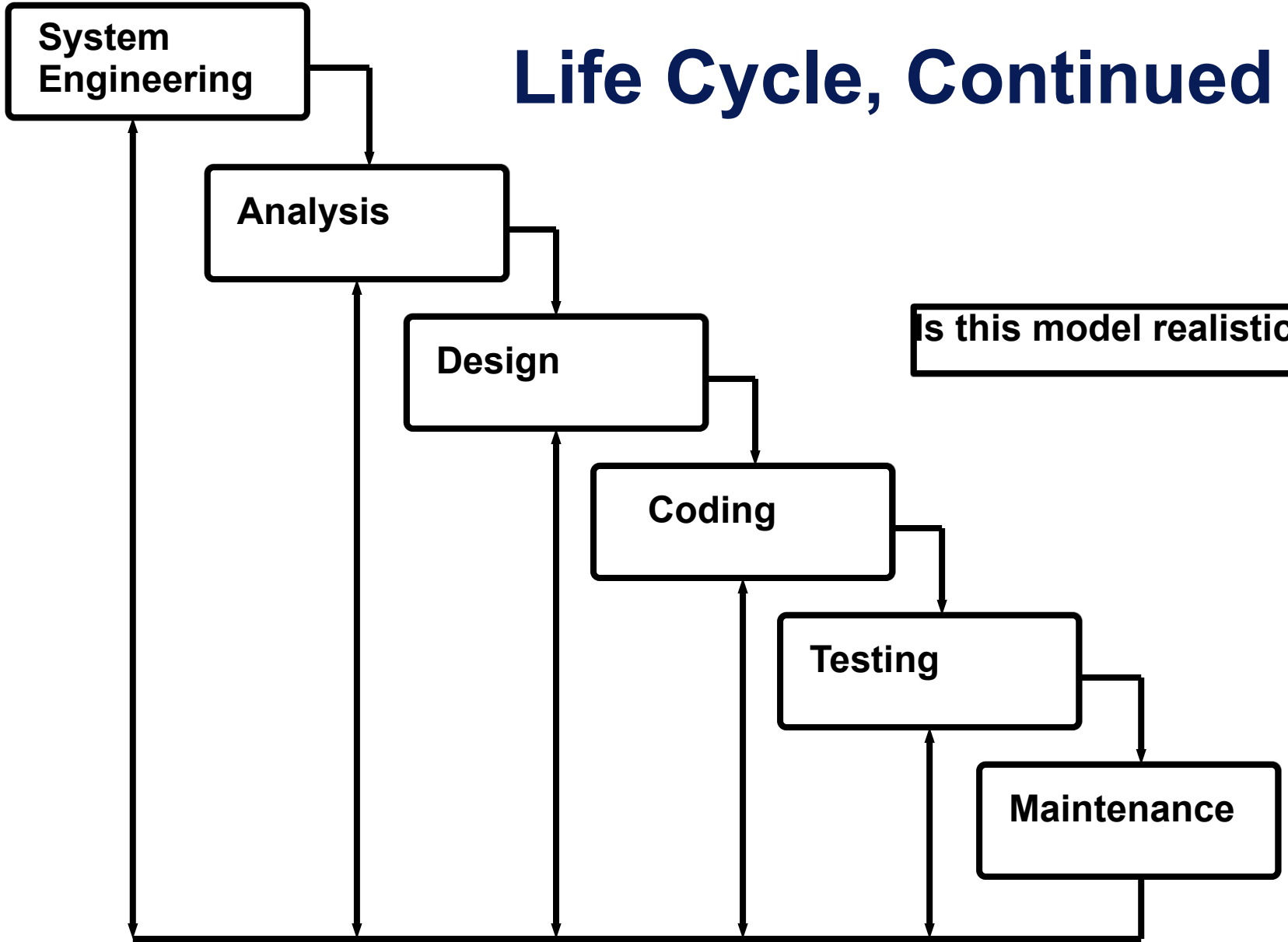
Testing

Maintenance

Life Cycle



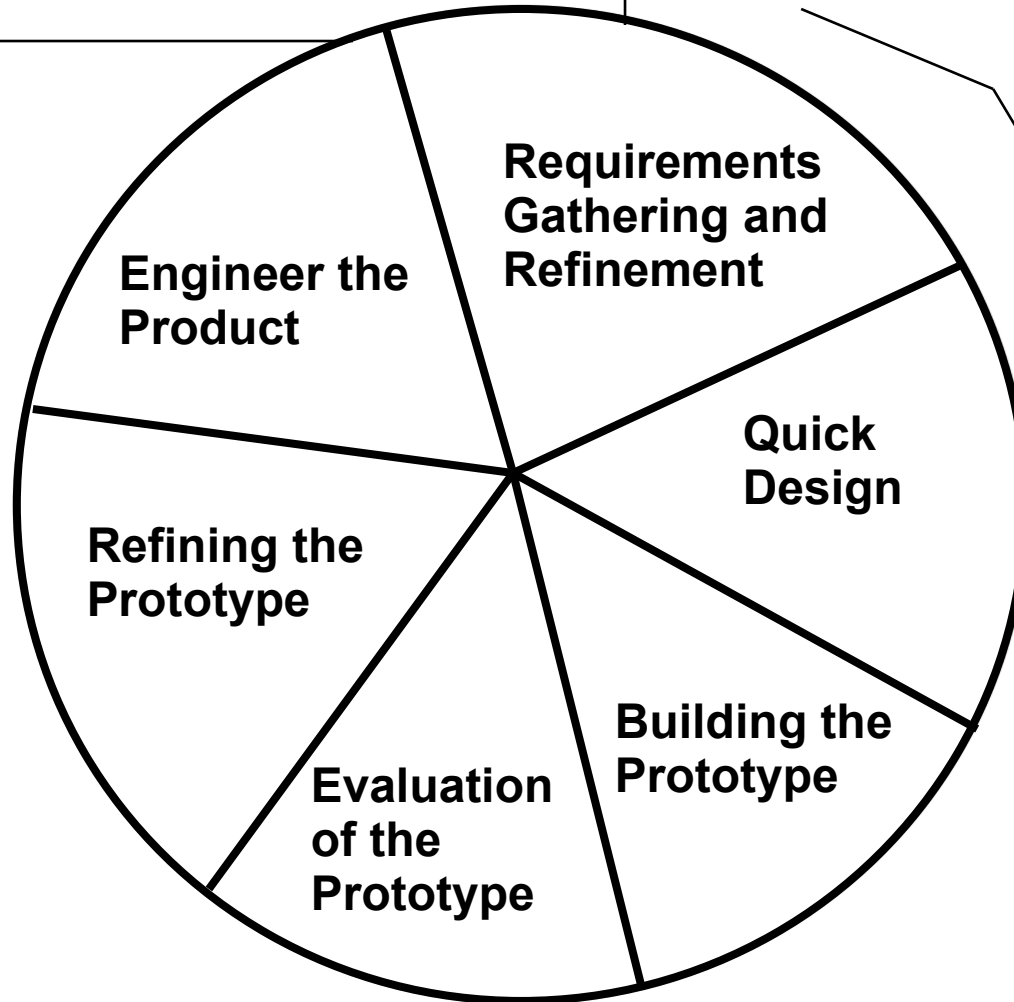
Life Cycle, Continued



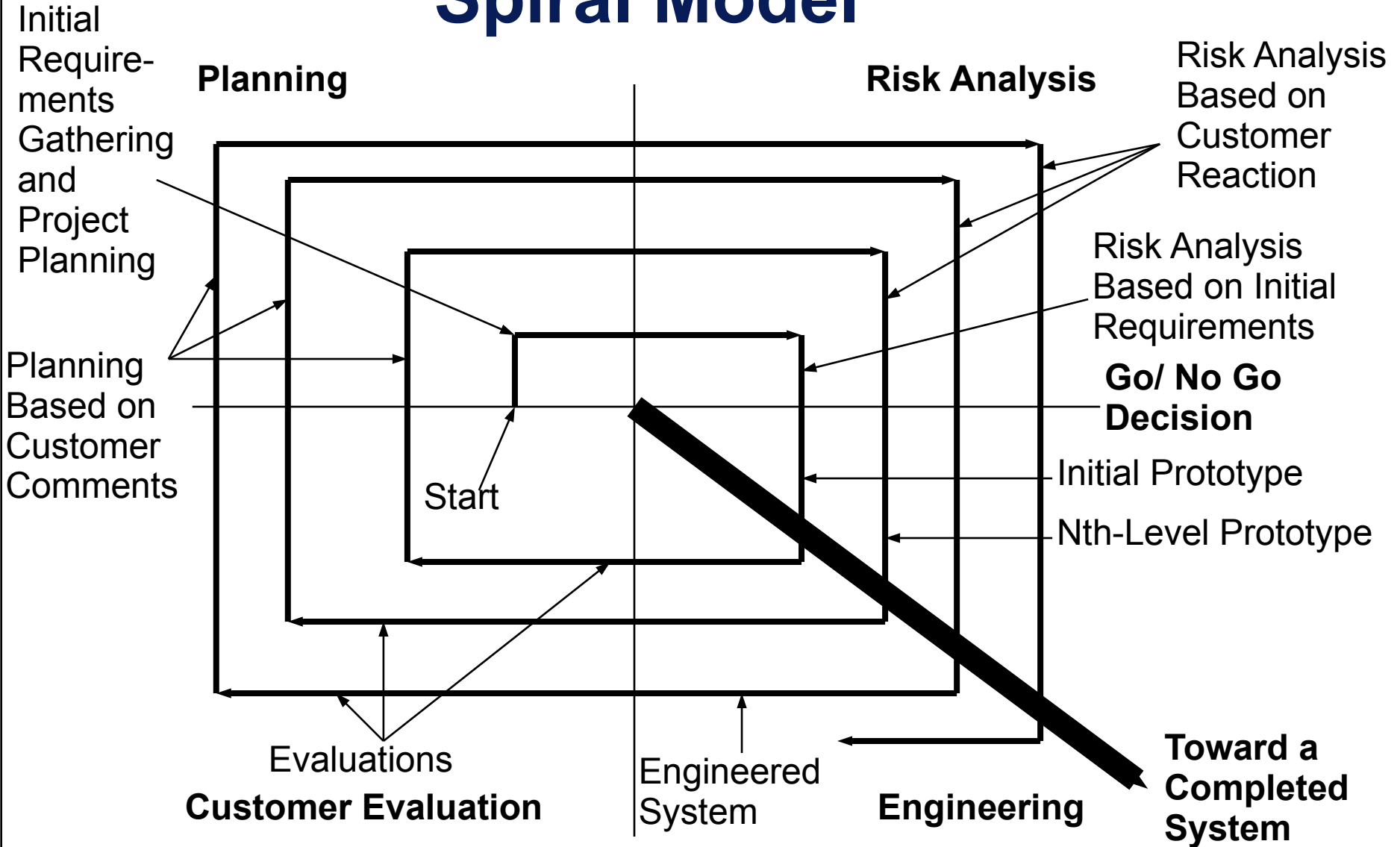
Is this model realistic?

Prototyping Model

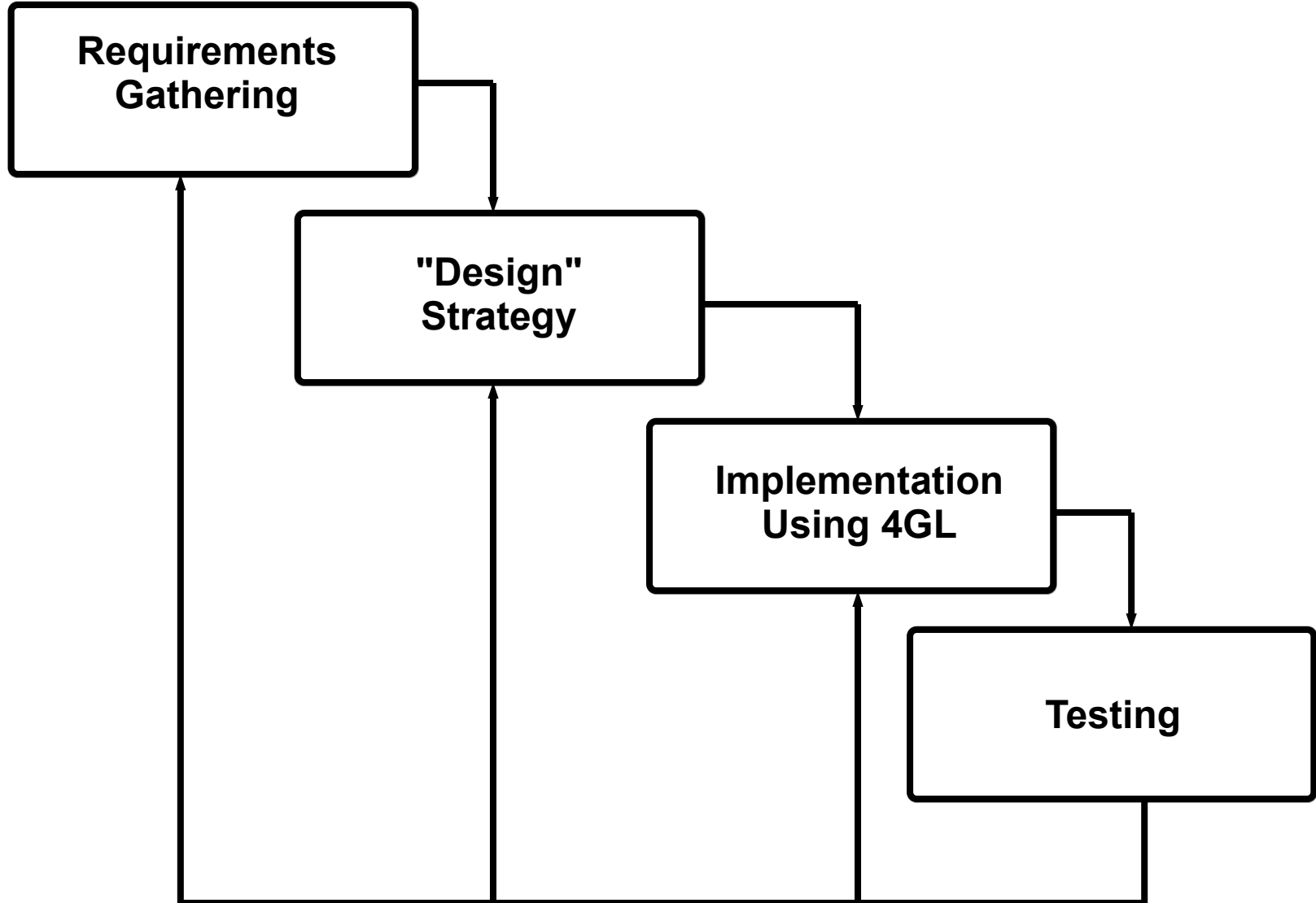
Start
Stop



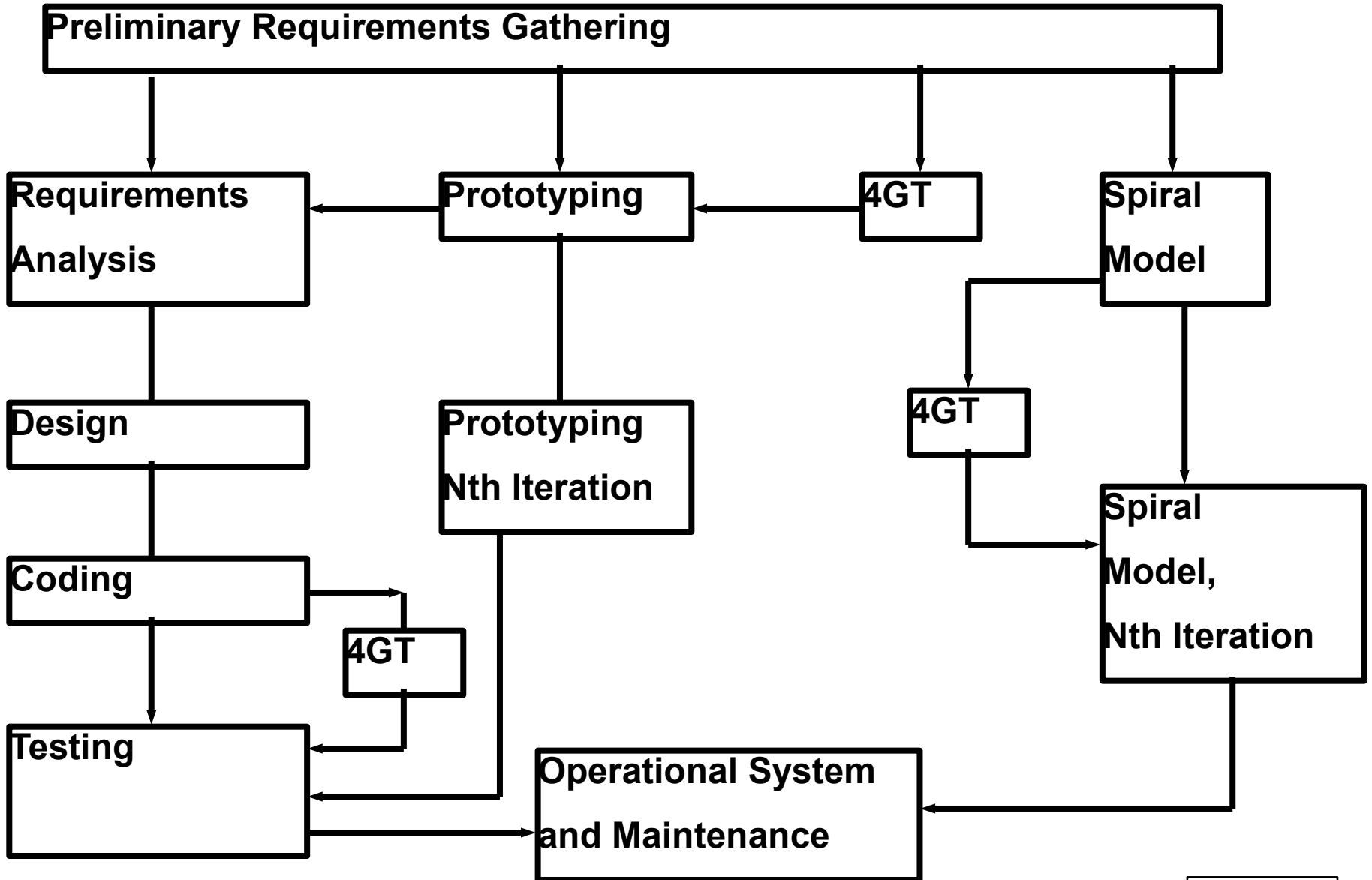
Spiral Model



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.

Increasing Process Maturity



Initial - Ad hoc;
unpredictable

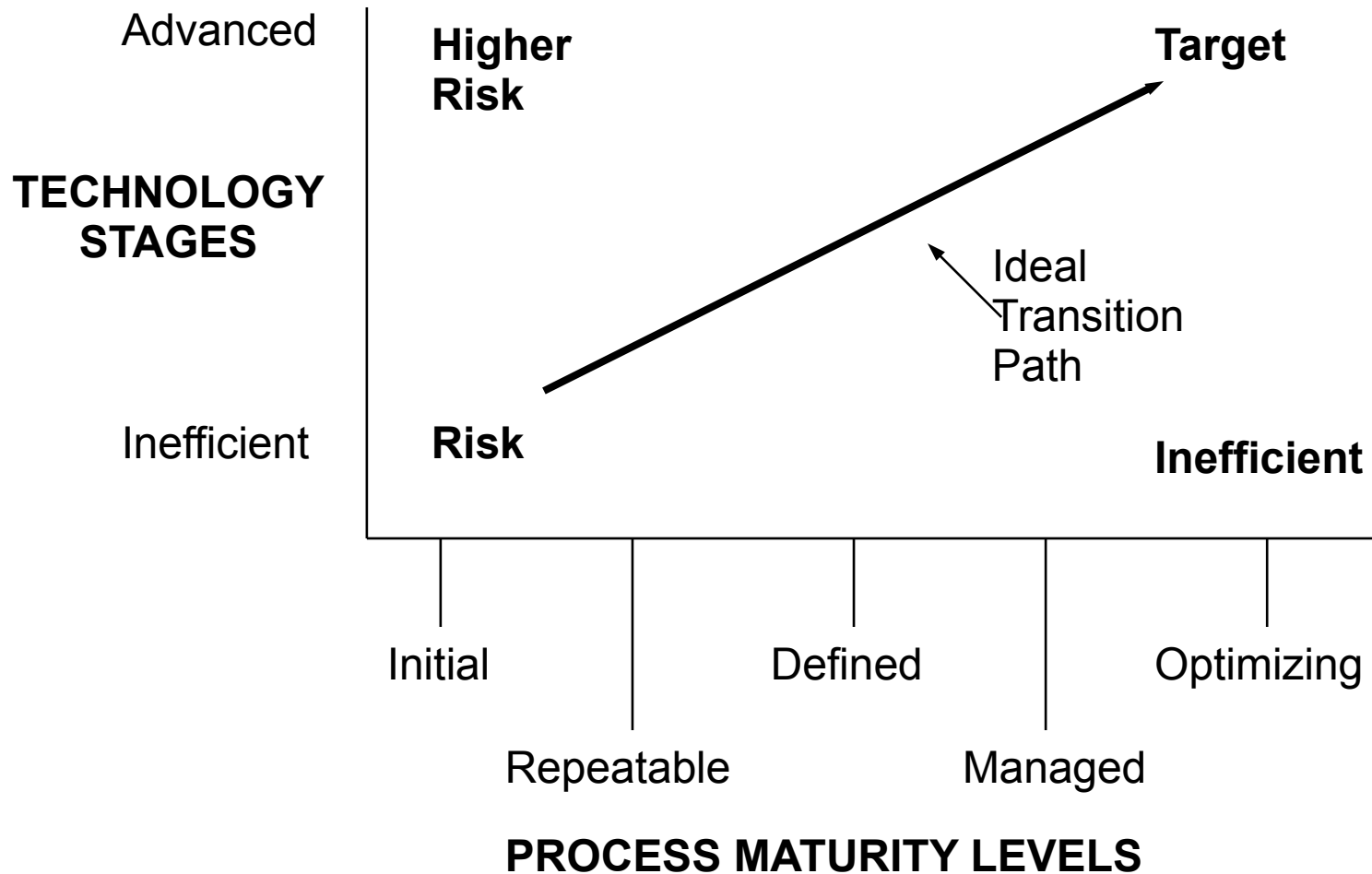
Repeatable - Costs,
Schedules managed

Defined - Process
institutionalized

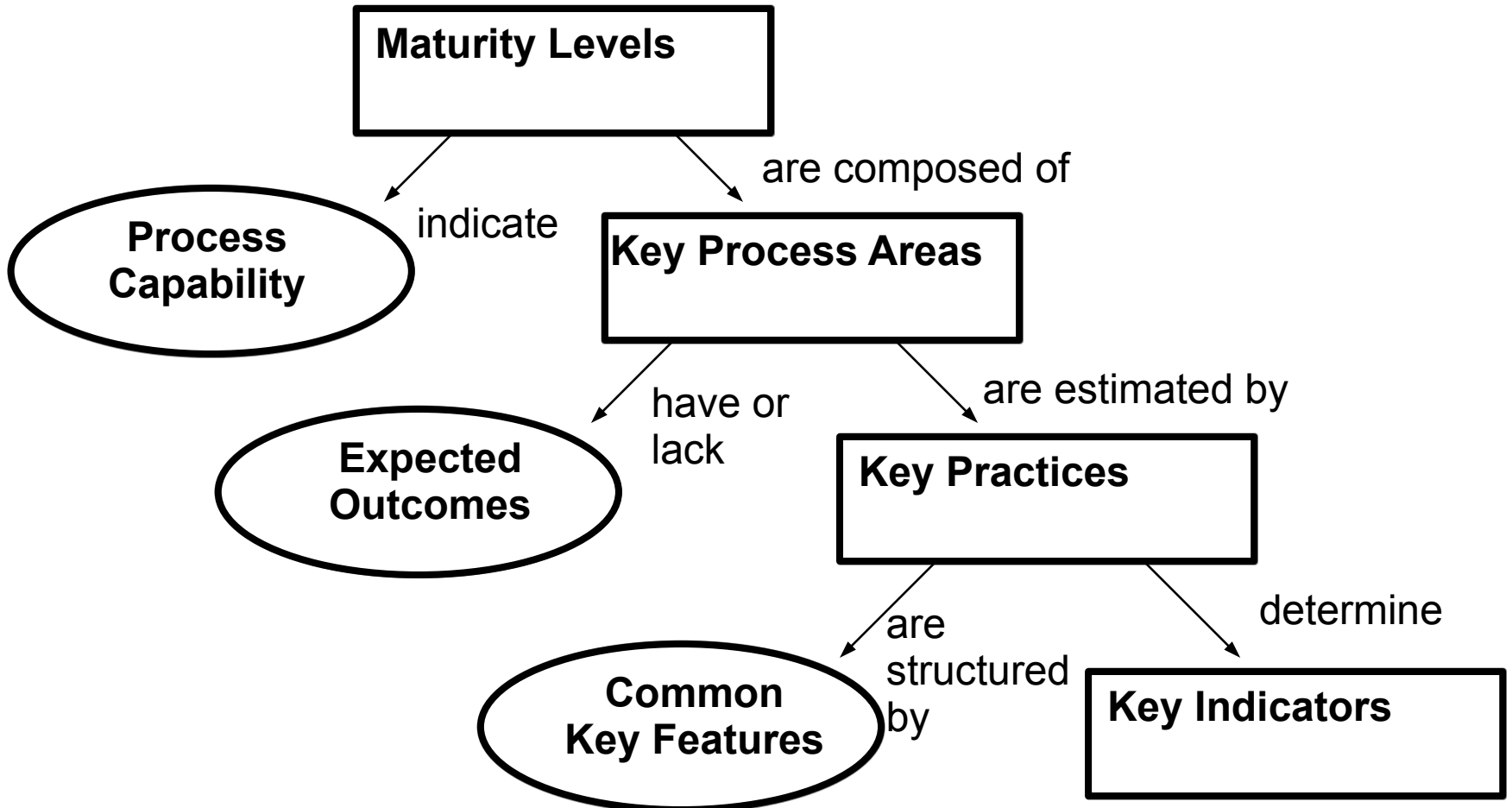
Managed - Process
measured/controlled

Optimizing - Process
refined constantly

Process Maturity and Technology



Maturity Keys



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, Continued

Ada
Specification



```
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, Continued

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.