TOPICS

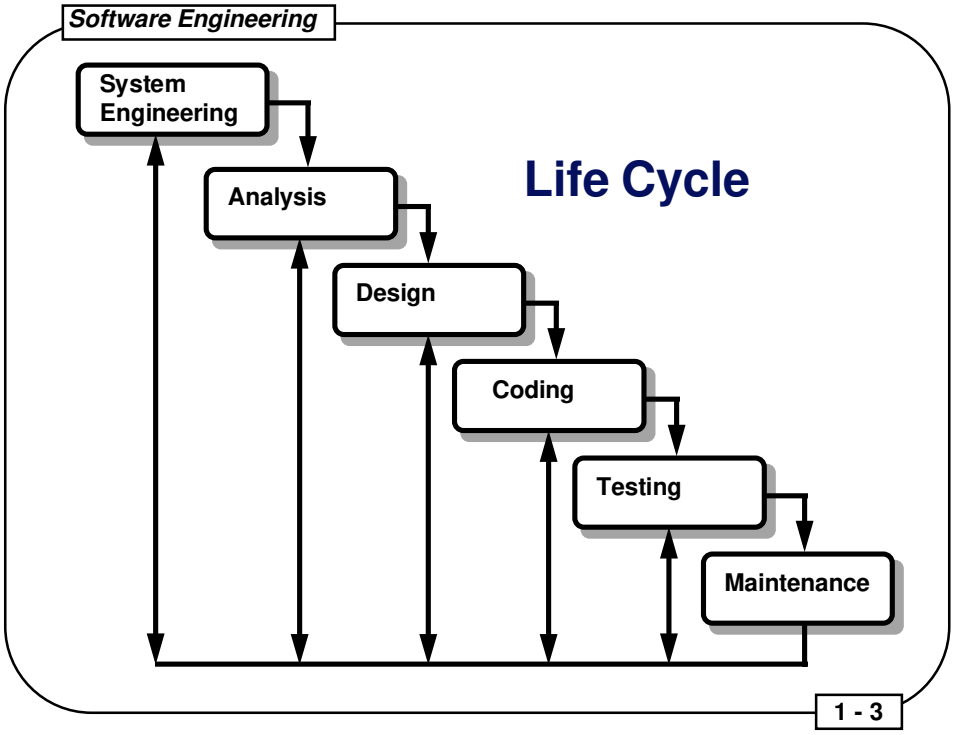
The Nature and History of Software Development

Problems with Software Development

Software Engineering Paradigms and Technology

SOFTWARE ENGINEERING PARADIGMS

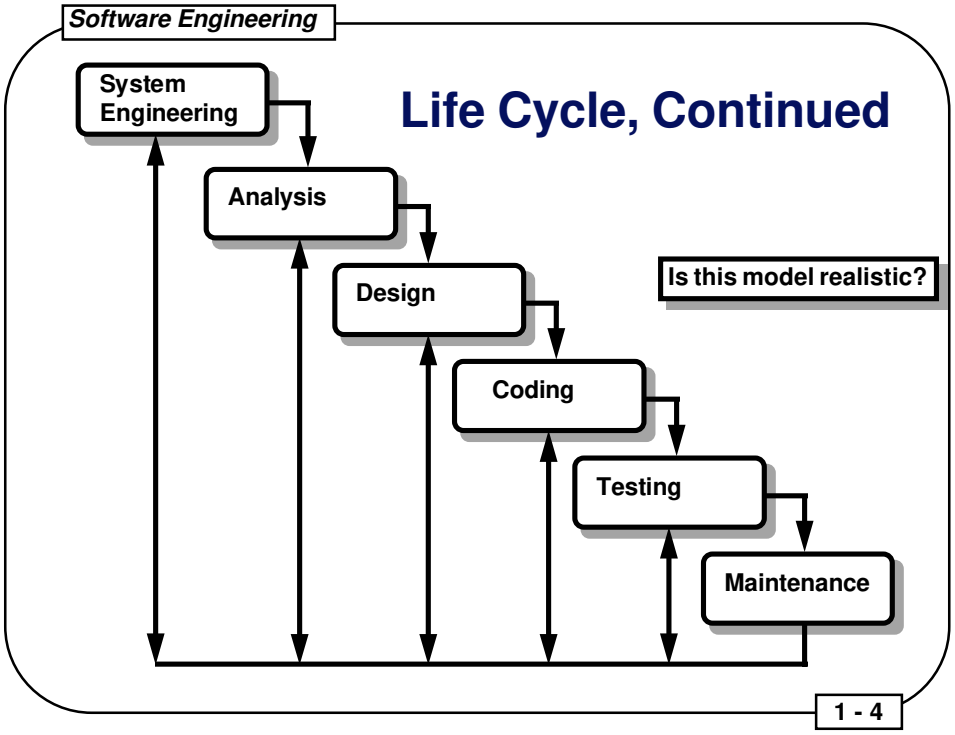
- Life Cycle
- Prototyping Model
- Spiral Model
- Fourth Generation Techniques
- Combining Paradigms
- Generic Paradigm



Classic "Waterfall" Model

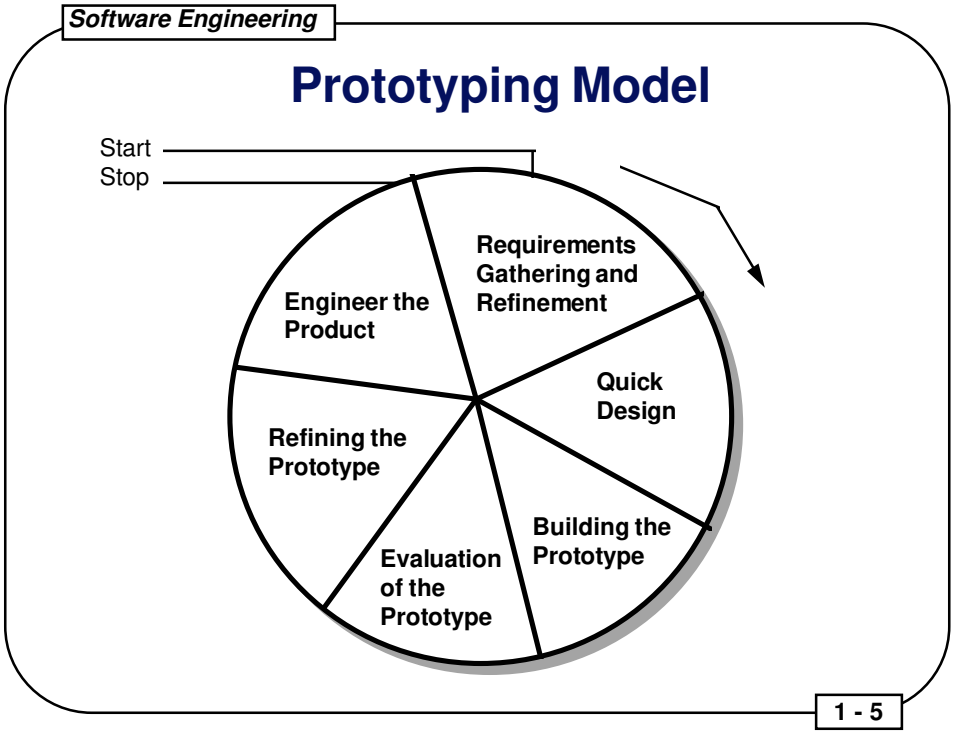
This model is a systematic, sequential approach to software development. It is the oldest and most often used of all software engineering paradigms.

- **System Engineering** - Establish requirements for the software as a part of the larger system. Determine which parts of the entire system are to be allocated to software.
- **Analysis** - Establish requirements from the point of view of the software. Include functional, performance, and interface requirements for the software subsystem.
- **Design** - Define the software architecture, procedural details, data structures, and interface characteristics for the software. The design process plans the implementation of the software to meet the requirements. Rapid prototyping and automated analysis of the design may come into play. The design of the software presents enough information so that a programmer who does not necessarily know how the system works can create code.
- **Coding** - The translation of a design into a compilable form. If the design is sufficiently detailed and adequate technologies are available, coding may be automatic.
- **Testing** - Analysis and verification that codes statements are fully compliant with the requirements and the customer's intent.
- **Maintenance** - The process of continuing to support the system after it is released to the customer. This process often involves several types of activities:
  - **Corrective Maintenance** - fixing errors
  - **Adaptive Maintenance** - changing the software to run in different environments (such as new versions of an OS or new target platforms)
  - **Enhancement** - adding new features to the software



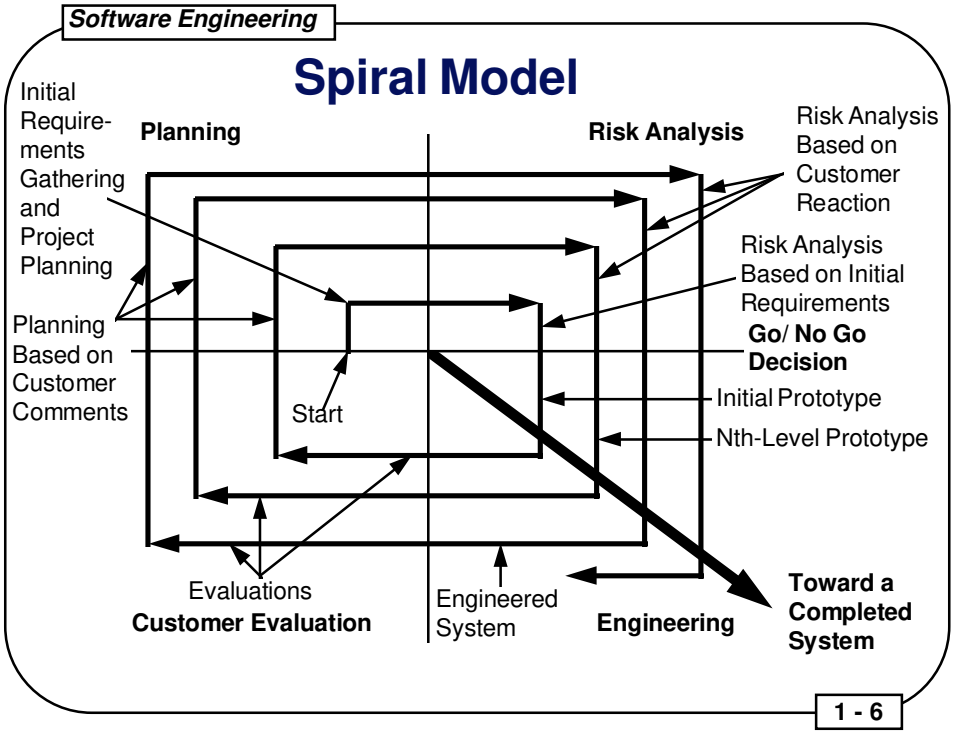
Problems with the Classic "Waterfall" Model

- Real projects rarely follow strict sequential development.
- It is very difficult to fully state all the requirements up front. The customer does not often know exactly what his requirements are or does not provide all the necessary input to fully state the requirements.
- This model demands patience from the customer. Working code is not available until very late into the project.



An Iterative Process

- **Requirements Gathering and Refinement** - During the first loop around this circle, an initial statement of the requirements is obtained. During later loops, the requirements statement is revised based on customer feedback.
- **Quick Design** - Very little time is usually spent on designing the prototype. Often, aided by workstation-based tools, we transition directly into building the prototype.
- **Building the Prototype** - This often involves the aid of software tools.
- **Evaluation of the Prototype** - The customer and the developers unite in their efforts to look at the prototype and determine its flaws.
- **Refining the Prototype** - This step is taken only if the prototype is not discarded.
- **Engineer the Product** - This step is taken when the customer and developer are completely satisfied.



Iterative Refinement

First Loop

- Start at the center of the spiral; plan the project and gather initial requirements
- Perform a risk analysis based on these initial requirements; make a go/no go decision; continue if go
- Create an initial prototype of the system
- Customer (and developer) evaluate the prototype

Second Loop

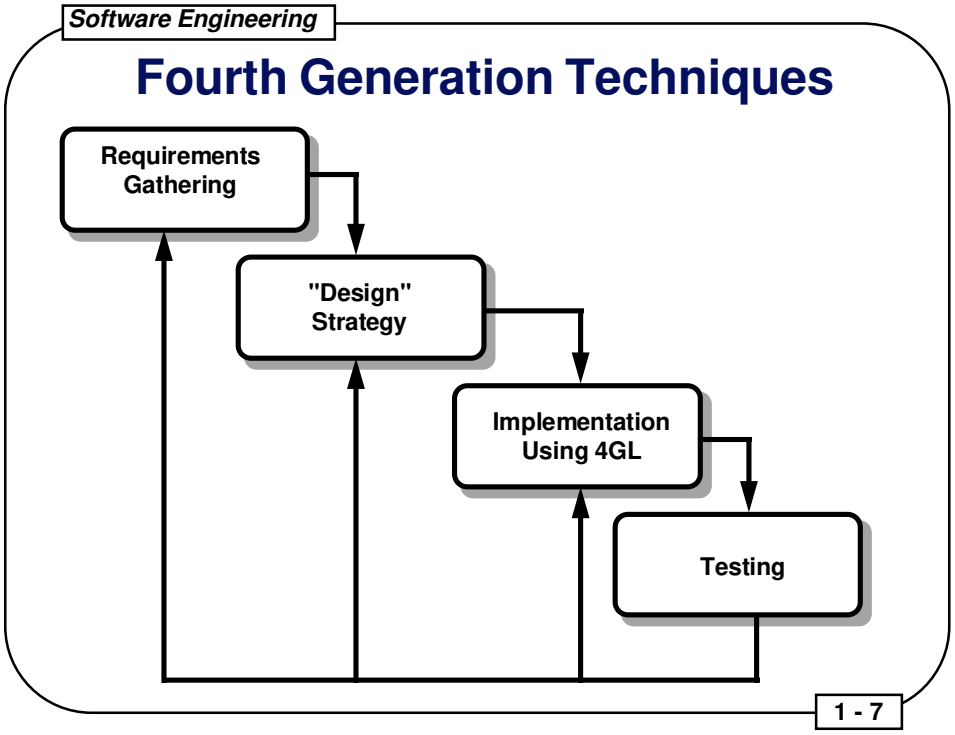
- Feedback from the evaluation is used to refine the requirements and more project planning is done
- Perform a second risk analysis based on the revised requirements; make a go/no go decision; continue if go
- Create a second prototype, based either on the initial prototype or built from scratch
- Customer (and developer) evaluate the second prototype

Nth Loop

- Repeat the Second Loop as desired

After Last Go/No Go Decision

- Engineer the system



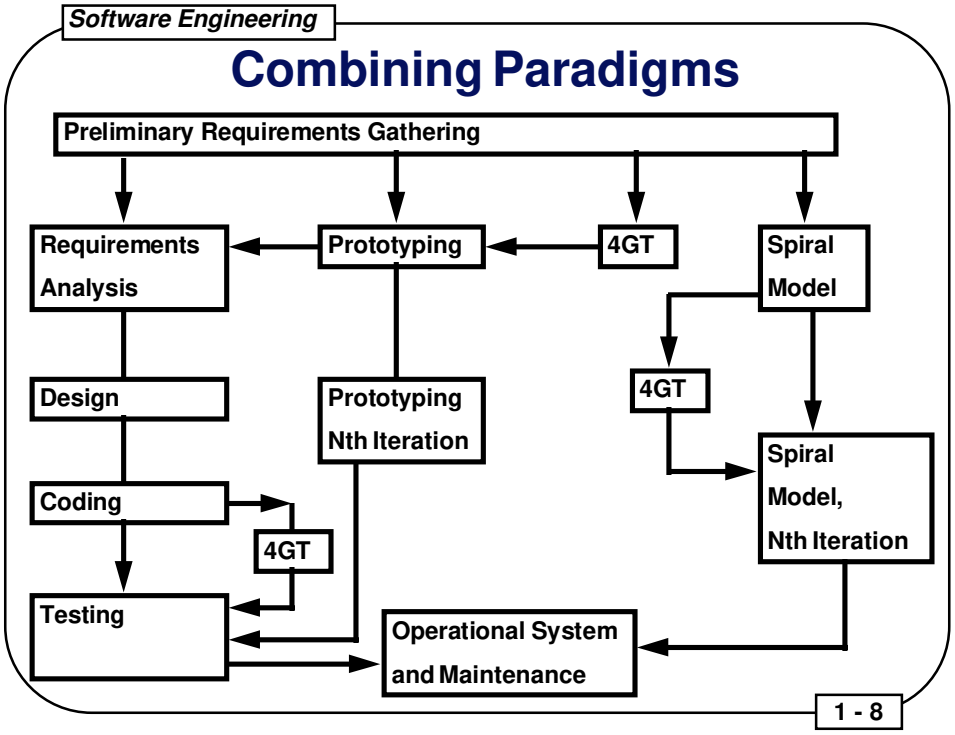
An Enhancement to the Classic "Waterfall"

First Pass

- Requirements Gathering - Collection of requirements as before
- "Design Strategy" - Design with 4th Generation Languages is often done online and is quite similar to coding
- Implementation Using 4GL - Coding
- Testing - Testing as before, with customer and developer evaluation

Iterations

- Reenter the waterfall where required as indicated by the evaluation



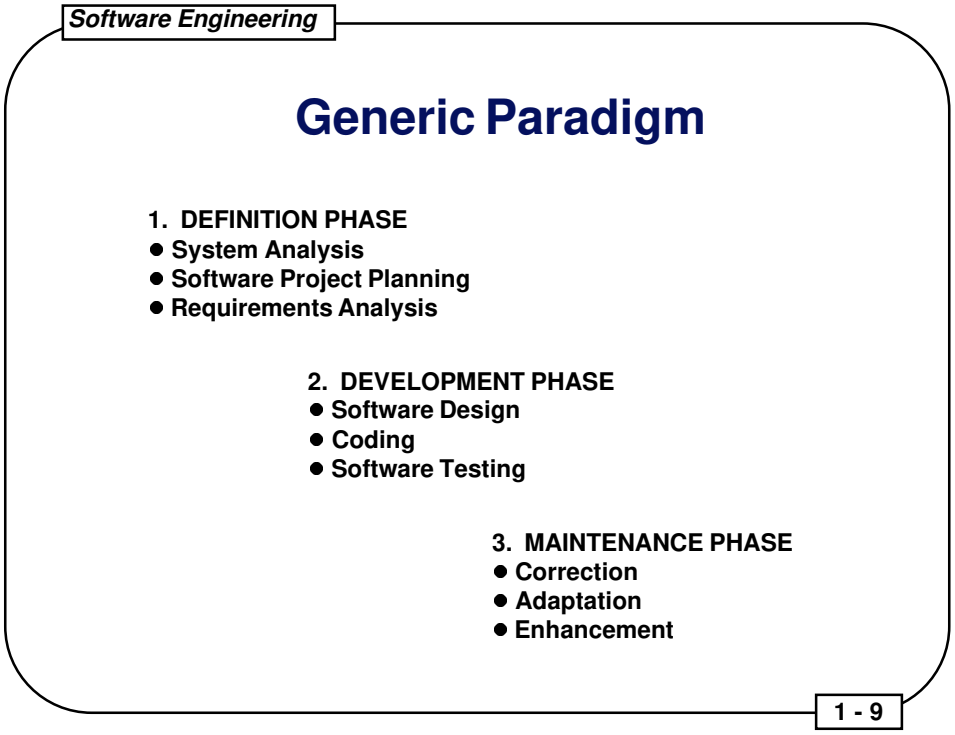
Applying Different Paradigms to Different Parts of the System

Preliminary Requirements Gathering

- A preliminary statement of requirements for the entire project is initially obtained
- Based on this statement of requirements, different methods are applied to different parts of the system as is deemed reasonable by the developer and the customer

Movement Through Each Method

- Different parts of the system are developed independently
- Integration may be a high risk area, so integration testing must be thorough



Common Phases for All Methods

Definition Phase

- All methods involve an analysis of the system in which the software resides, the gathering of the requirements for the software, and the planning of the development of the software
- Plan the development and get an initial understanding of the requirements

Development Phase

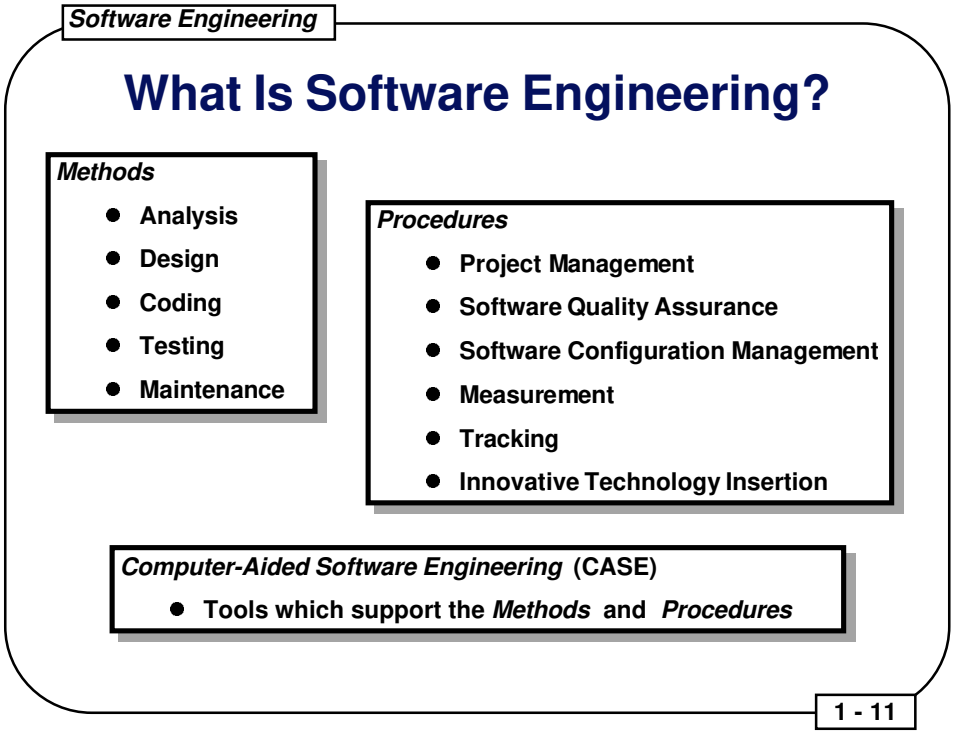
- Design, code, and test the software

Maintenance Phase

- Support the software after it is released to the customer; there are often three kinds of maintenance to be performed:
  - **Corrective Maintenance** - fix defects uncovered in the software
  - **Adaptive Maintenance** - change the software to run under different environments, such as new versions of an operating system
  - **Enhancement** - extend the capabilities of the software

SOFTWARE ENGINEERING TECHNOLOGY

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



The Essence of Software Engineering

Methods

- *Methods* comprise the techniques used to perform the various phases of the software development
- *Methods* are not necessarily documented formally and are often unique to each organization and its culture
- Once a method is selected for a project, automated facilities may come into play to support the method; a common flaw in many organizations is that automated facilities are sometimes selected first and people then spend time figuring out how to apply the facility to their methods or adapt them methods to the facility

Procedures

- *Procedures* are formal, documented activities performed during the various phases of the software development
- Personnel with less advanced training are often employed in roles which implement the various procedures
- Implementation of the procedures is one of the best places to apply automated techniques

CASE Tools

- *Computer-Aided Software Engineering tools* can be a valuable aid when applied to support a well-established method or set of methods
- CASE tools can also introduce a high degree of risk to a project if the organization is immature in its methods

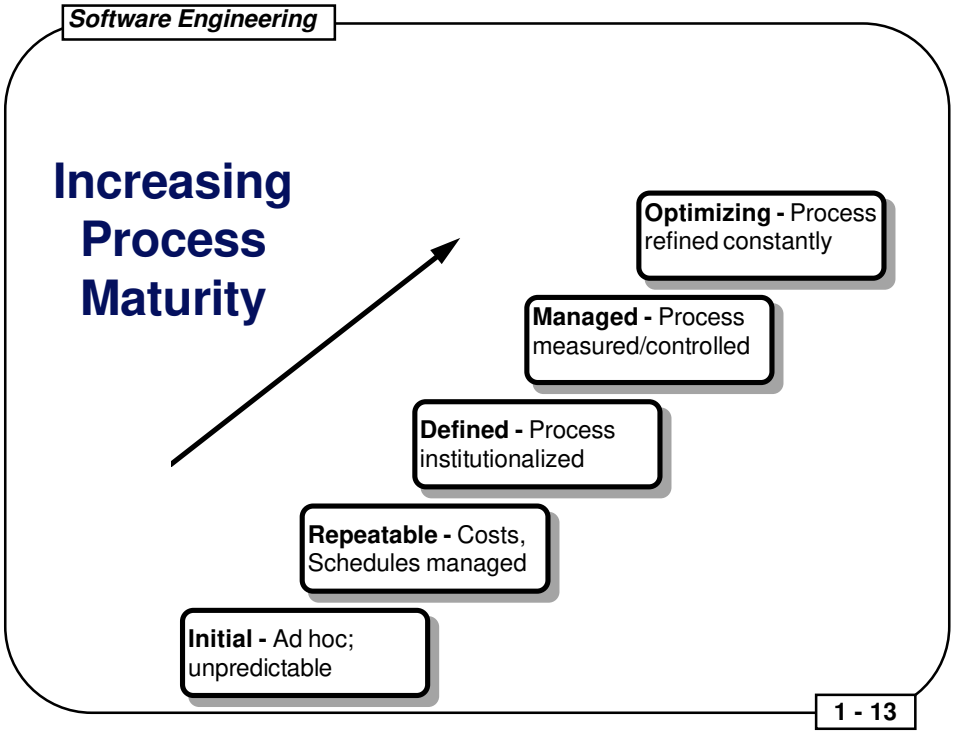
Software Engineering Capability and Its Measurement

- **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.**

Capability Maturity Model for Software

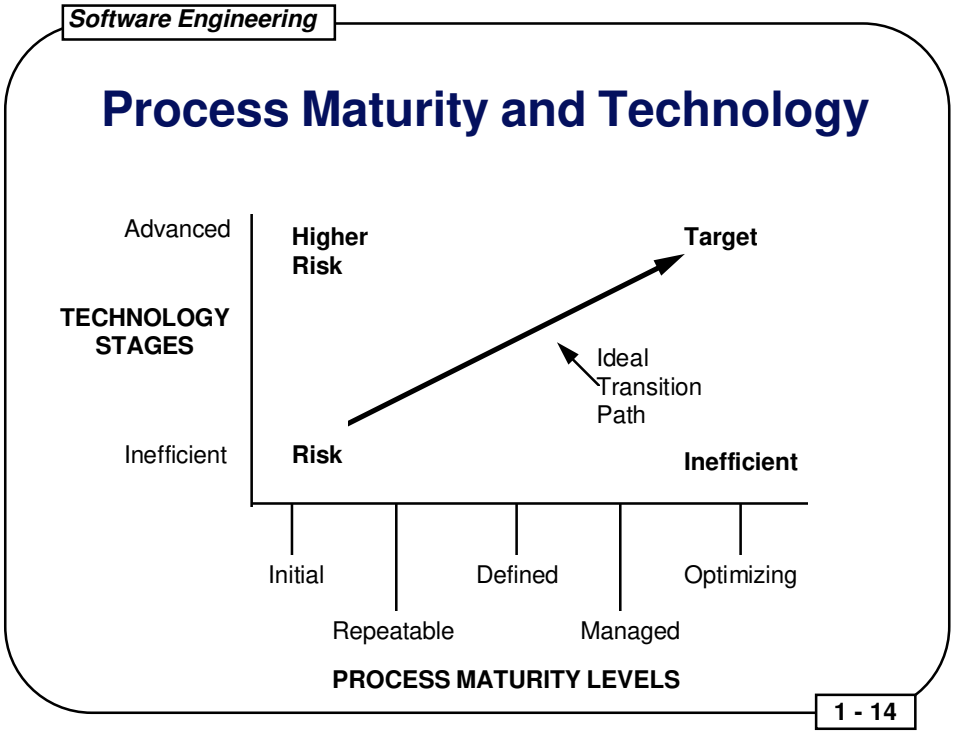
This model is defined in two papers from the Software Engineering Institute:

- Paulk, Curtis, Chrissis, *et al*, **Capability Maturity Model for Software**, August, 1991, Report Number CMU/SEI-91-TR-24 and ESD-TR-91-24, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213
- Weber, Paulk, Wise, Withey, *et al*, **Key Practices of the Capability Maturity Model**, August, 1991, Report Number CMU/SEI-91-TR-25 and ESD-TR-91-25, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213



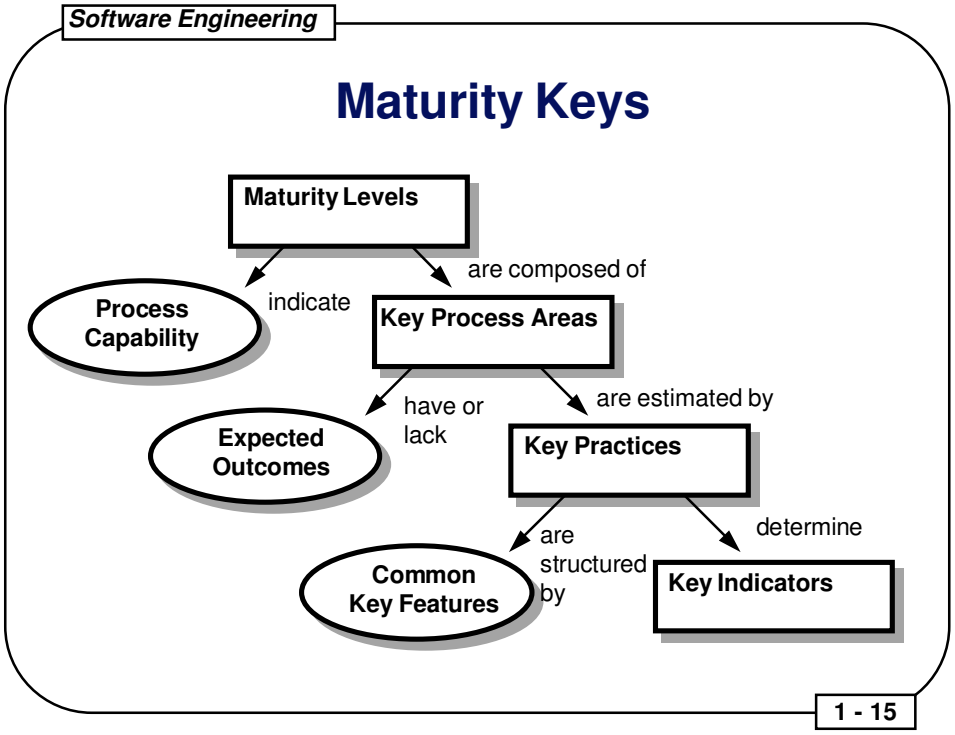
Some Aspects of Each Level

- Level 1: **Initial**
  - Project outcomes are characterized by frequent cost and schedule overruns
  - People are burnt out in the attempt to meet the schedule
- Level 2: **Repeatable**
  - Controls, software quality assurance, and baseline management are in place
  - No commitments are made without thorough review
  - Given experience with one type of project, probability of repeating the level of performance (cost, schedule, and quality) on another similar project is high
- Level 3: **Defined**
  - Process for each project is defined in writing at the outset
  - SQA monitors compliance with standards and is empowered to intervene
  - Project outcomes become more predictable across a broader range of projects
- Level 4: **Managed**
  - Quantitative quality and productivity goals are set for each step in the process
  - High predictability is achieved for each step of the process
- Level 5: **Optimizing**
  - Data collected are used to identify weakness and bottlenecks in the process
  - Causes of errors are analyzed, and future errors prevented



State of the Practice

- Corporations in the United States (based on SEI Corporate Affiliates who voluntarily took the CMM evaluation in 1989) --
  - 74% are at Level 1 (Initial)
  - 22% are at Level 2 (Repeatable)
  - 4% are at Level 3 (Defined)
- Corporations in Japan (based on a visit to Japan by an SEI team in 1990) --
  - 95%+ are at Level 1 (Initial)
  - 5%- are at Level 2 (Repeatable)



Common Key Features of the Key Practices include --

- Goals
- Commitment to Perform
- Ability to Perform
- Activities Performed
- Monitoring Implementation
- Verifying Implementation

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

Goals for Key Process Areas in Level 2

- **Requirements Management**
  - The system requirements allocated to software provide a clearly stated, verifiable, and testable foundation for software engineering and software management.
  - The allocated requirements define the scope of the software effort.
  - The allocated requirements and changes to the allocated requirements are incorporated into the software plans, products, and activities in an orderly manner.
- **Software Project Planning**
  - A plan is developed that appropriately and realistically covers the software activities and commitments.
  - All affected groups and individuals understand the software estimates and plans and commit to support them.
  - The software estimates and plans are documented for use in tracking the software activities and commitments.

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

Goals for Key Process Areas in Level 2

- **Software Project Tracking and Oversight**
  - Actual results and performance of the software project are tracked against documented and approved plans.
  - Corrective actions are taken when the actual results and performance of the software project deviate significantly from the plans.
  - Changes to software commitments are understood and agreed to by all affected groups and individuals.
- **Software Subcontract Management**
  - The prime contractor selected qualified subcontractors.
  - The software standards, procedures, and product requirements for the subcontract comply with the prime contractor's commitments.
  - Commitments between the prime contractor and subcontractor are understood and agreed to by both parties.
  - The prime contractor tracks the subcontractor's actual results and performance against the commitments.

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**

Goals for Key Process Areas in Level 2

- **Software Quality Assurance**
  - Compliance of the software product and software process with applicable standards, procedures, and product requirements is independently confirmed.
  - When there are compliance problems, management is aware of them.
  - Senior management addresses noncompliance issues.
- **Software Configuration Management**
  - Controlled and stable baselines are established for planning, managing, and building the system.
  - The integrity of the system's configuration is controlled over time.
  - The status and content of the software baselines are known.

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

Goals for Key Process Areas in Level 3

- **Organization Process Focus**
  - Current strengths and weaknesses of the organization's software process are understood and plans are established to systematically address the weaknesses.
  - A group is established with appropriate knowledge, skills, and resources to define a standard software process for the organization.
  - The organization provides the resources and support needed to record and analyze the use of the organization's standard software process in order to maintain and improve it.
- **Organization Process Definition**
  - A standard software process for the organization is defined and maintained as a basis for stabilizing, analyzing, and improving the performance of the software projects.
  - Specifications of common software processes and documented process experiences from past and current projects are collected and available.

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

Goals for Key Process Areas in Level 3

- **Training Program**
  - The staff and managers have the skills and knowledge to perform their jobs.
  - The staff and managers effectively use, or are prepared to use, the capabilities and features of the existing and planned work environment.
  - The staff and managers are provided with opportunities to improve their professional skills.
- **Integrated Software Management**
  - The planning and managing of each software project is based on the organization's standard software process.
  - Technical and management data from past and current projects are available and used to effectively and efficiently estimate, plan, track, and replan the software projects.

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

Goals for Key Process Areas in Level 3

- **Software Product Engineering**
  - Software engineering issues for the product and the process are properly addressed in the system requirements and system design.
  - The software engineering activities are well-defined, integrated, and used consistently to produce a software system.
  - State-of-the-practice software engineering tools and methods are used, as appropriate, to build and maintain the software system.
- **Intergroup Coordination**
  - The project's technical goals and objectives are understood and agreed to by its staff and managers.
  - The responsibilities assigned to each of the project groups and the working interfaces between these groups are known to all groups.
  - The project groups are appropriately involved in intergroup activities and in identifying, tracking, and addressing intergroup issues.
  - The project groups work as a team.

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**

Goals for Key Process Areas in Level 3

- **Peer Reviews**
  - Product defects are identified and fixed early in the life cycle.
  - Appropriate product improvements are identified and implemented early in the life cycle.
  - The staff members become more effective through a better understanding of their work products and knowledge of errors that can be prevented.
  - A rigorous group process for reviewing and evaluating product quality is established and used.

Key Process Areas by Level
Level 4 (Managed)

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

Goals for Key Process Areas in Level 4

- **Process Measurement and Analysis**
  - The organization's standard software process is stable and under statistical process control.
  - The relationship between product quality, productivity, and product development cycle time is understood in quantitative terms.
  - Special causes of process variation (i.e., variations attributable to specific applications of the process and not inherent in the process) are identified and controlled.
- **Quality Management**
  - Measurable goals and priorities for product quality are established and maintained for each software project through interaction with the customer, end users, and project groups.
  - Measurable goals for process quality are established for all groups involved in the software process.
  - The software plans, design, and process ar adjusted to bring forecasted process and product quality in line with the goals.
  - Process measurements are used to manage the software project quantitatively.

Key Process Areas by Level
Level 5 (Optimizing)

- Defect Prevention
- Technology Innovation
- Process Change Management

Goals for Key Process Areas in Level 5

- **Defect Prevention**
  - Sources of product defects that are inherent or repeatedly occur in the software process activities are identified and eliminated.
- **Technology Innovation**
  - The organization has a software process and technology capability to allow it to develop or capitalize on the best available technologies in the industry.
  - Selection and transfer of new technology into the organization is orderly and thorough.
  - Technology innovations are tied to quality and productivity improvements of the organization's standard software process.
- **Process Change Management**
  - The organization's staff and managers are actively involved in setting quantitative, measurable improvements goals and in improving the software process.
  - The organization's standard software process and the projects' defined software processes continually improve.
  - The organization's staff and managers are able to use the evolving software processes and their supporting tools and methods properly and effectively.

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.**

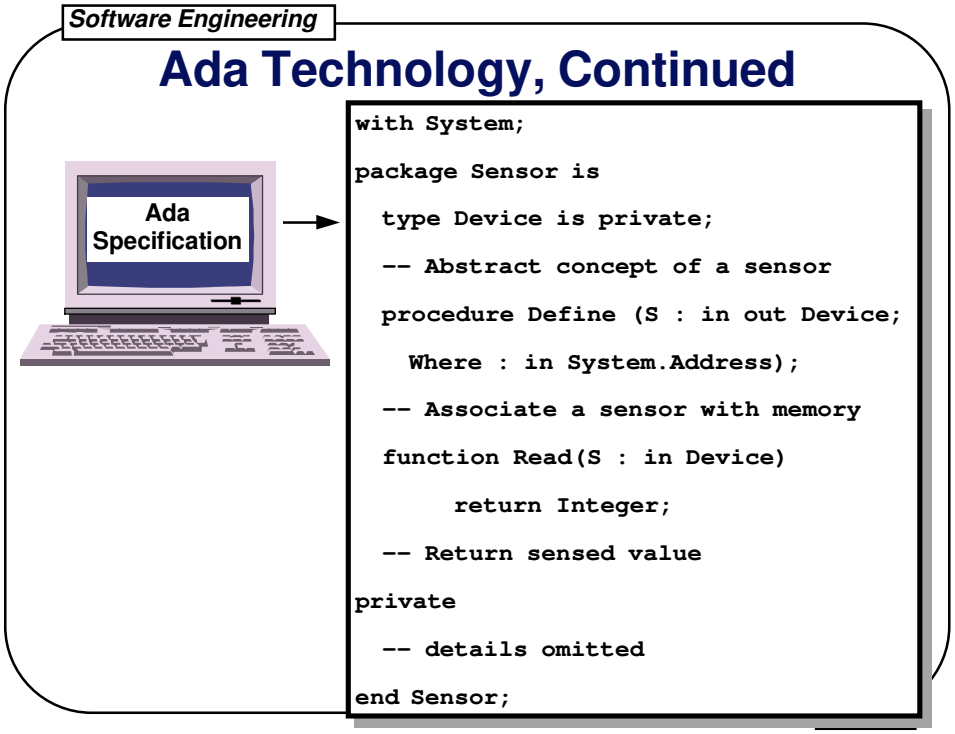
Ada

The Ada programming language is defined in detail in:

ANSI/MIL-STD-1815A, **Ada Programming Language**, 22 January 1983, United States Department of Defense, Under Secretary for Defense, Research, and Engineering

To further understand the reasoning behind the choices made in the design of Ada, the following document is highly recommended:

Ichbiah, Barnes, Firth, Woodger, **Rationale for the Design of the Ada Programming Language**, 1986, Honeywell Systems and Research Center, MN65-2100, 3660 Technology Drive, Minneapolis, MN 55418 and Alsys, 29 Avenue de Versailles, 78170 La Celle Saint Cloud, France



Ada Code Example

This is an example of an Ada package which defines a class of objects of type **Sensor.Device.**

This code example is incomplete. The details of the private section of the package specification are omitted, and the package body, in which the procedure **Define** and the function **Read** are implemented, is not shown.

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.

Suggested Reading

- Grady Booch, **Object Oriented Design with Applications**, 1991, The Benjamin/Cummings Publishing Company, Inc., ISBN 0-8053-0091-0
- Grady Booch, **Software Components with Ada**, 1987, The Benjamin/Cummings Publishing Company, Inc., ISBN 0-8053-0610-2
- Grady Booch, **Software Engineering with Ada**, 1987, The Benjamin/Cummings Publishing Company, Inc., ISBN 0-8053-0604-8
- R.J.A. Buhr, **System Design with Ada**, 1984, Prentice-Hall, Inc., ISBN 0-13-881623-9
- David Naiditch, **Rendezvous with Ada: A Programmer's Introduction**, 1989, John Wiley & Sons, ISBN 0-471-61654-0
- The Software Productivity Consortium, **Ada Quality and Style: Guidelines for Professional Programmers**, 1989, Van Nostrand Reinhold, ISBN 0-442-23805-3