OOSE
Chapter 1

INTRODUCTION

Chapter 1 - Introduction

1.1 Software Engineering

Layered Technology

Application

Process Framework

Process Flow

CMMI

Software Engineering

The quest of attempting answer to following questions:

- Why does it take so long to get software finished?
- Why are development costs so high?
- Why can’t we find all errors before we give the software to our customers?
- Why do we spend so much time and effort maintaining existing programs?
- Why do we continue to have difficulty in measuring progress as software is being developed and maintained?

Software Engineering

- These, and many other questions, are a manifestation of the concern about software and the manner in which it is developed—a concern that has lead to the adoption of software engineering practice.

What is Software?

Textbook description of software might take the following form:

Software is:

(1) instructions (computer programs) that when executed provide desired features, function, and performance;

(2) data structures that enable the programs to adequately manipulate information, and

(3) descriptive information in both hard copy and virtual forms that describes the operation and use of the programs.

What differentiate software from hardware?

- *Software is developed or engineered; it is not manufactured in the classical sense.*
- *Software doesn’t “wear out.”*
- *Although the industry is moving toward component-based construction, most software continues to be custom built.*

Software Application Domain

- **System software**—a collection of programs written to service other programs. Some system software
- e.g., compilers, editors, and file management Utilities
- processes complex, but determinate information structures.

Software Application Domain

- **Engineering/scientific software**—has been characterized by “number crunching” algorithms.

- Applications range from astronomy to volcanology, from automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing.

Software Application Domain

- **Application software**—stand-alone programs that solve a specific business need.

- Applications in this area process business or technical data in a way that facilitates business operations or management/technical decision making.

Software Application Domain

- **Embedded software**—resides within a product or system and is used to implement and control features and functions for the end user and for the system itself.

- Embedded software can perform limited and esoteric functions (e.g., key pad control for a microwave oven) or provide significant function and control capability (e.g., digital functions in an automobile such as fuel control, dashboard displays, and braking systems).

Software Application Domain

- **Product-line software**—designed to provide a specific capability for use by many different customers.

- Product-line software can focus on a limited and esoteric marketplace (e.g., inventory control products) or address mass consumer markets (e.g., word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, and personal and business financial applications).

Software Application Domain

- **Web applications**—called “WebApps,” this network-centric software category spans a wide array of applications. In their simplest form, WebApps can be little more than a set of linked hypertext files that present information using text and limited graphics.
- However, as Web 2.0 emerges, WebApps are evolving into sophisticated computing environments that not only provide stand-alone features, computing functions, and content to the end user, but also are integrated with corporate databases and business applications.

Software Application Domain

- **Artificial intelligence software**—makes use of non numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis.

- Applications within this area include robotics, expert systems, pattern recognition (image and voice), artificial neural networks, theorem proving, and game playing.

Challenges for Software development

- Software has become deeply embedded in virtually every aspect of our lives, and as a consequence, the number of people who have an interest in the features and functions provided by a specific application has grown dramatically.

- *It follows that a concerted effort should be made to understand the problem before a software solution is developed.*

Challenges for Software development

- The complexity of these new computer-based systems and products demands careful attention to the interactions of all system elements.

- *It follows that design becomes a pivotal activity.*

Challenges for Software development

- If the software fails, people and major enterprises can experience anything from minor inconvenience to catastrophic failures.

- *It follows that software should exhibit high quality.*

Challenges for Software development

- As the perceived value of a specific application grows, the likelihood is that its user base and longevity will also grow.

  As its user base and time-in-use increase, demands for adaptation and enhancement will also grow.

- *It follows that software should be maintainable.*

Software Engineering

- These lead to one conclusion: *software in all of its forms and across all of its application domains should be engineered.*

- And that leads us to the topic —*software engineering.*

Software Engineering

- Although hundreds of authors have developed personal definitions of software engineering, a definition proposed by **Fritz Bauer [Nau69]** at the seminal conference on the subject still serves as a basis for discussion:

- [Software engineering is] the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.

Software Engineering

- The IEEE [IEE93a] has developed a more comprehensive definition when it states:

Software Engineering: (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1)

Frequently asked Questions

- What is software?

- Computer programs and associated documentation.
- Software products may be developed for a particular customer or may be developed for a general market.

Frequently asked Questions

- What are the attributes of good software?

- Good software should deliver the required functionality and performance to the user and should be maintainable, dependable, and usable.

Frequently asked Questions

- What is software engineering?

- Software engineering is an engineering discipline that is concerned with all aspects of software production.

Frequently asked Questions

- What are the fundamental software engineering activities?

- Software specification, software development, software validation, and software evolution.

Frequently asked Questions

- What is the difference between software engineering and computer science?

- Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.

Frequently asked Questions

- What is the difference between software engineering and system engineering?

System engineering is concerned with all aspects of computer-based systems development including hardware, software, and process engineering.

Software engineering is part of this more general process.

Frequently asked Questions

- What are the key challenges facing software engineering?

- Coping with increasing diversity,
- demands for reduced delivery times, and
- developing trustworthy software

Frequently asked Questions

- What are the costs of software engineering?

- Roughly 60% of software costs are development costs; 40% are testing costs.
- For custom software, evolution costs often exceed development costs.

Frequently asked Questions

- What are the best software engineering techniques and methods?

- While all software projects have to be professionally managed and developed, different techniques are appropriate for different types of system.
- For example, games should always be developed using a series of prototypes whereas safety critical control systems require a complete and analyzable specification to be developed.
- You can’t, therefore, say that one method is better than another.

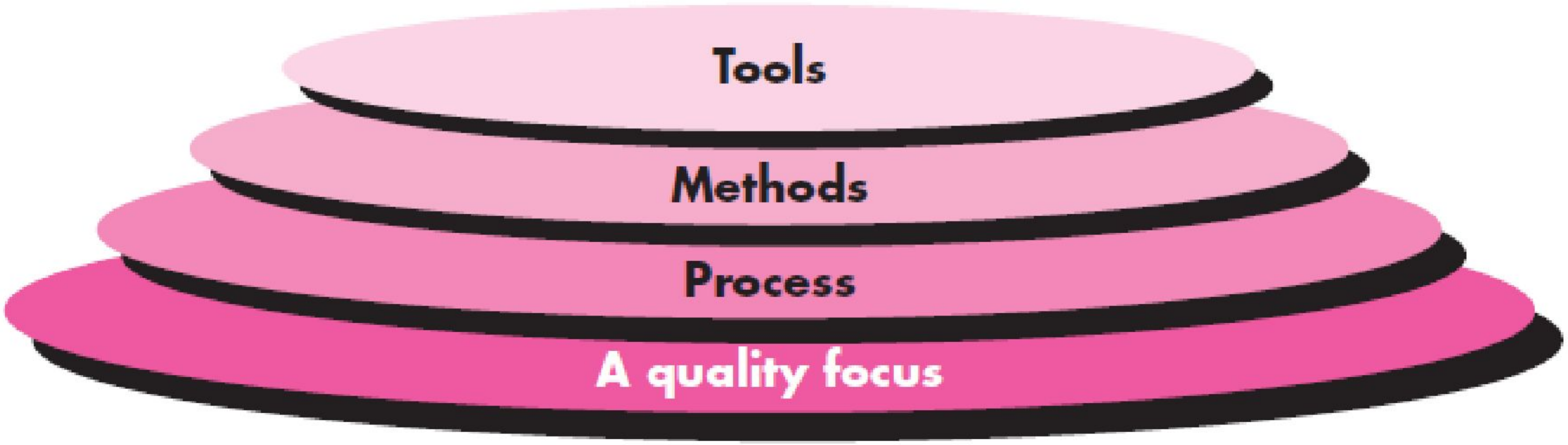
Frequently asked Questions

- What differences has the Web made to software engineering?

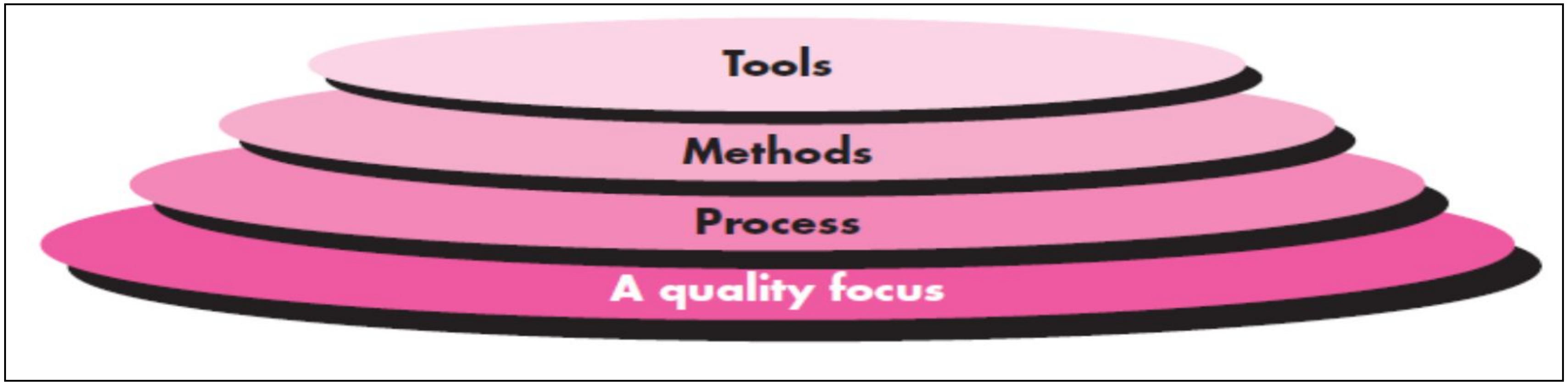
- The Web has led to the availability of software services and the possibility of developing highly distributed service-based systems.
- Web-based systems development has led to important advances in programming languages and software reuse.

Layered Technology

- Software engineering is a layered technology.

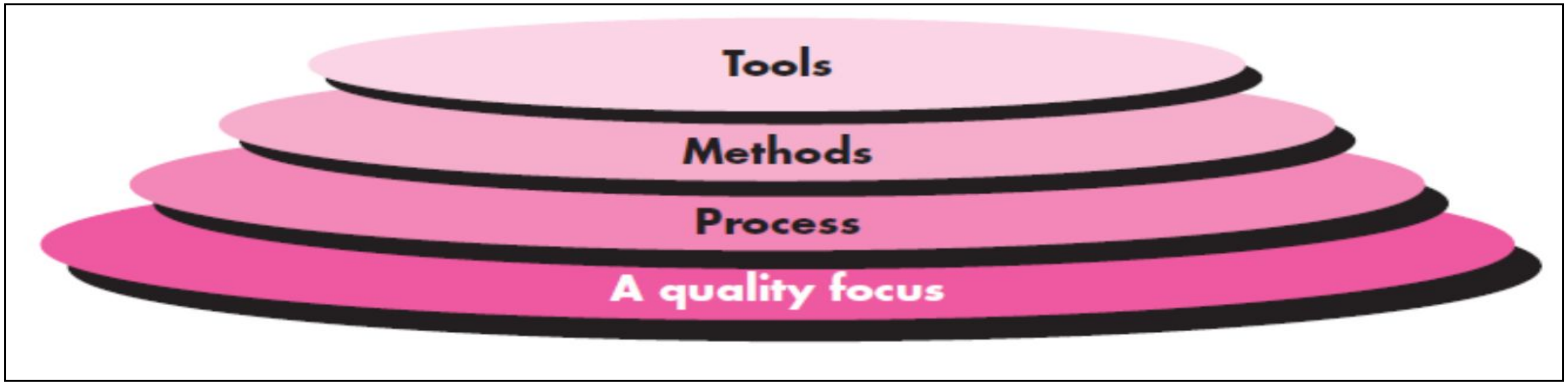


Layered Technology



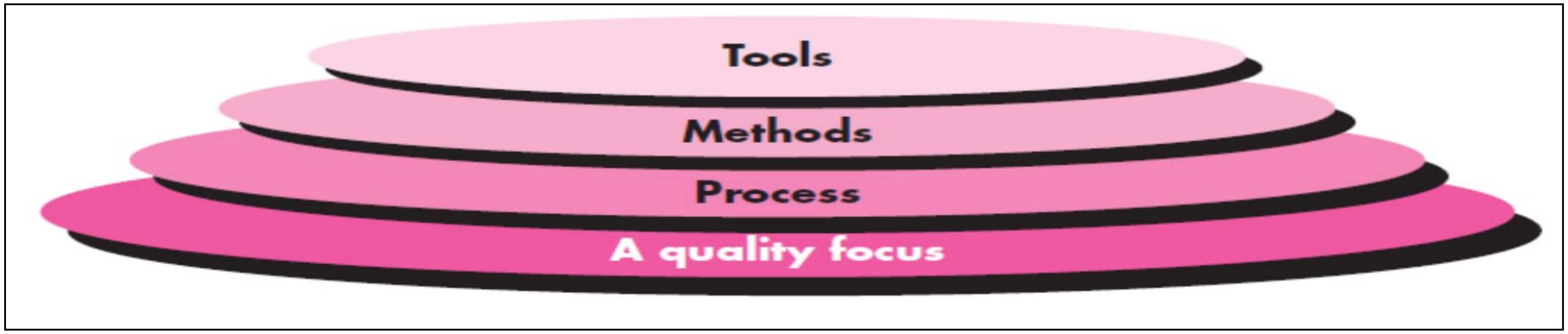
- Quality Focus:
- Any engineering approach (including software engineering) must rest on an organizational commitment to quality.
- Total quality management, Six Sigma, and similar philosophies10
- foster a continuous process improvement culture, and it is this culture that ultimately leads to the development of increasingly more effective approaches to software engineering.
- The bedrock that supports software engineering is a quality focus.

Layered Technology



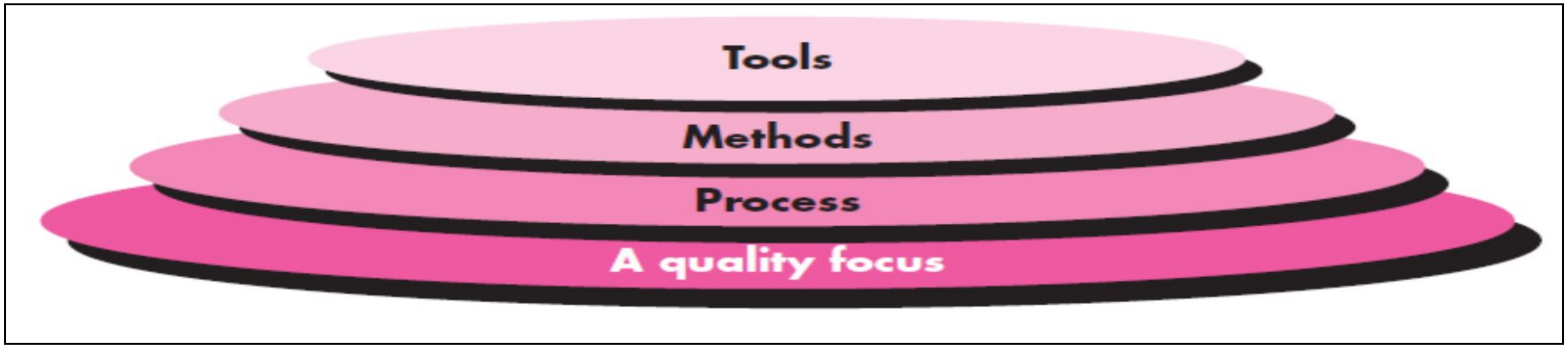
- Process:
- The foundation for software engineering is the *process* layer.
- The software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software.
- Process defines a framework that must be established for effective delivery of software engineering technology.

Layered Technology



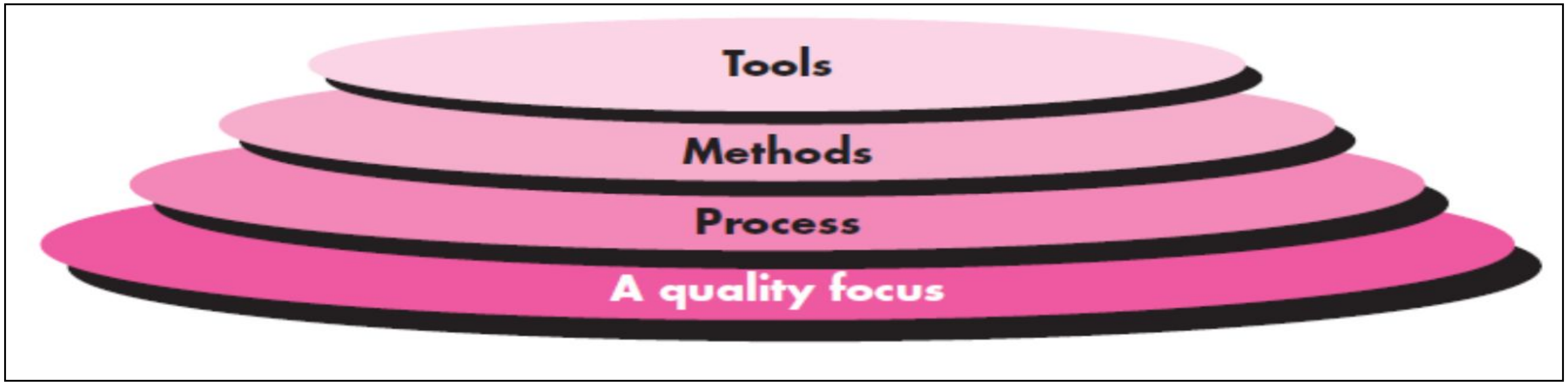
- The software process forms the basis for
  - management control of software projects and
  - establishes the context in which technical methods are applied, work products (models, documents, data, reports, forms, etc.) are produced,
  - milestones are established,
  - quality is ensured, and
  - change is properly managed.

Layered Technology



- Methods:
- Software engineering *methods* provide the technical how-to's for building software.
- Methods encompass a broad *array of tasks that include communication, requirements analysis, design modeling, program construction, testing, and support*.
- Software engineering methods rely on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques.

Layered Technology



- Tools:
- Software engineering *tools* provide automated or semi automated support for the process and the methods.

- When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called *computer-aided software engineering*, is established.

Application of Software Engineering

- **There are software engineering fundamentals that apply to all types of software system:**

1. They should be developed using a managed and understood development
process. The organization developing the software should plan the development
process and have clear ideas of what will be produced and when it will be completed.

2. Dependability and performance are important for all types of systems. Software should behave as expected, without failures and should be available for use when it is required. It should be safe in its operation and, as far as possible,
should be secure against external attack. The system should perform efficiently.

Application of Software Engineering

- **There are software engineering fundamentals that apply to all types of software system:**

3. Understanding and managing the software specification and requirements (what the software should do) are important. You have to know what different customers and users of the system expect from it and you have to manage their expectations so that a useful system can be delivered within budget and to schedule.

4. You should make as effective use as possible of existing resources and should not waste resources. This means that, where appropriate, you should reuse software that has already been developed rather than write new software.

Process Framework

- A generic process framework for software engineering defines five framework activities—**communication, planning, modelling, construction,** and **deployment.**

Process Framework

- **Communication**

- *Who has a stake in the solution to the problem?* That is, who are the stakeholders?
- *What are the unknowns?* What data, functions, and features are required to properly solve the problem?
- *Can the problem be compartmentalized?* Is it possible to represent smaller problems that may be easier to understand?
- *Can the problem be represented graphically?* Can an analysis model be created?

Process Framework

- Planning

- *Have you seen similar problems before?* Are there patterns that are recognizable in a potential solution? Is there existing software that implements the data, functions, and features that are required?
- *Has a similar problem been solved?* If so, are elements of the solution reusable?
- *Can subproblems be defined?* If so, are solutions readily apparent for the subproblems?
- *Can you represent a solution in a manner that leads to effective implementation?*

Process Framework

- Modelling
- *Does the solution conform to the plan?* Is source code traceable to the design model?
- *Is each component part of the solution provably correct?* Have the design and code been reviewed, or better, have correctness proofs been applied to the algorithm?

Process Framework

- *Is it possible to test each component part of the solution?* Has a reasonable testing strategy been implemented?

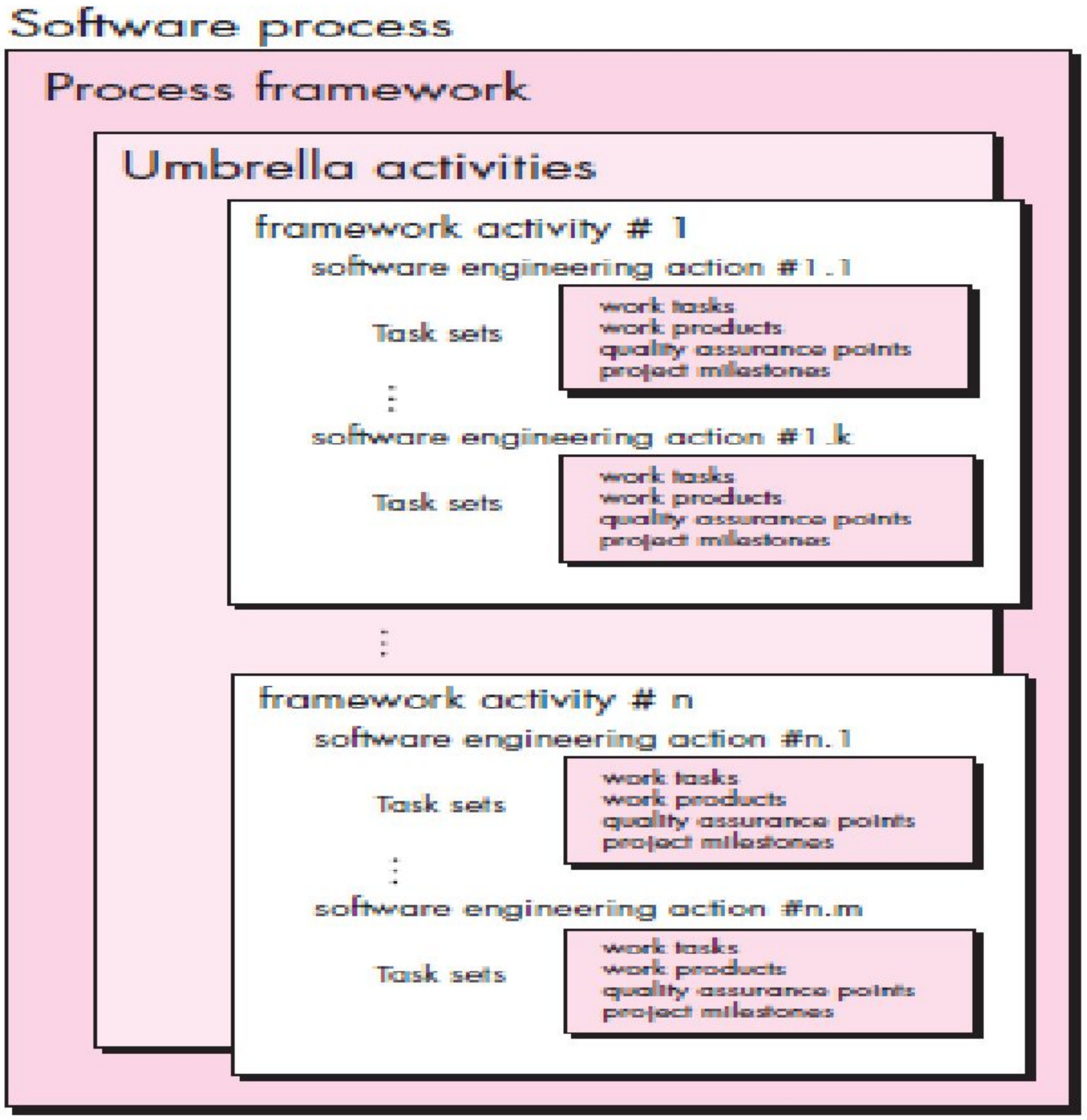
- *Does the solution produce results that conform to the data, functions, and features that are required?* Has the software been validated against all stakeholder requirements?

Process Framework

- A generic process framework for software engineering defines five framework activities—**communication, planning, modeling, construction**, and **deployment.**

- In addition, a set of umbrella activities**—project tracking and control, risk management, quality assurance, configuration management, technical reviews, and others**—are applied throughout th

Process Framework



Process Framework

- Defining a Framework Activity

- *What actions are appropriate for a framework activity, given the nature of the problem to be solved, the characteristics of the people doing the work, and the stakeholders who are sponsoring the project?*

Process Framework

- Identifying a Task Set

- Each software engineering action can be represented by a number of different *task sets*—each a collection of software engineering work tasks, related work products, quality assurance points, and project milestones.

Process Framework

- Task Set

- A task set defines the actual work to be done to accomplish the objectives of a software engineering action.
- For example, *elicitation* (more commonly called "requirements gathering") is an important software engineering action that occurs during the communication activity.
- The goal of requirements gathering is to understand what various stakeholders want from the software that is to be built.

Process Framework

- Process Patterns

- A *process pattern* describes a process-related problem that is encountered during software engineering work, identifies the environment in which the problem has been encountered, and suggests one or more proven solutions to the problem.

Process Framework- Example

- For a small software project requested by one person (at a remote location) with simple, straightforward requirements, the communication activity might encompass little more than **a phone call** with the appropriate stakeholder.
- Therefore, the only necessary action is *phone conversation,* and the work tasks (the *task set*) that this action encompasses are: 1. Make contact with stakeholder via telephone.

  2. Discuss requirements and take notes.

  3. Organize notes into a brief written statement of requirements.

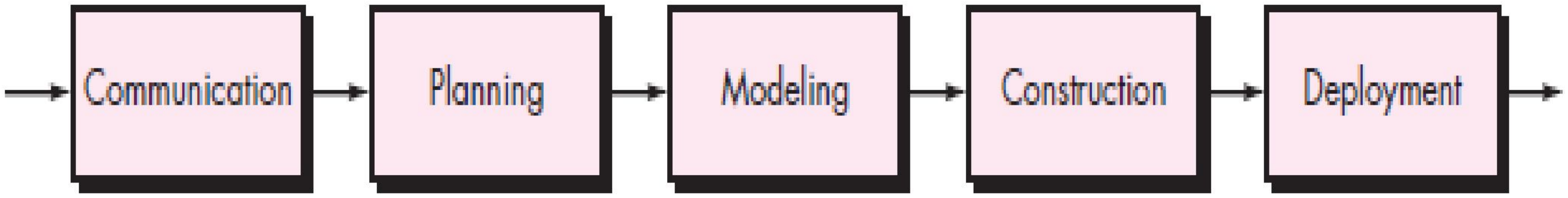
  4. E-mail to stakeholder for review and approval.

Process Flow

- *process flow*—describes how the framework activities and the actions and tasks that occur within each framework activity are organized with respect to sequence and time.
- Different types of process flow are:
  - Linear
  - Iterative
  - evolutionary
  - Parallel

Linear Process flow

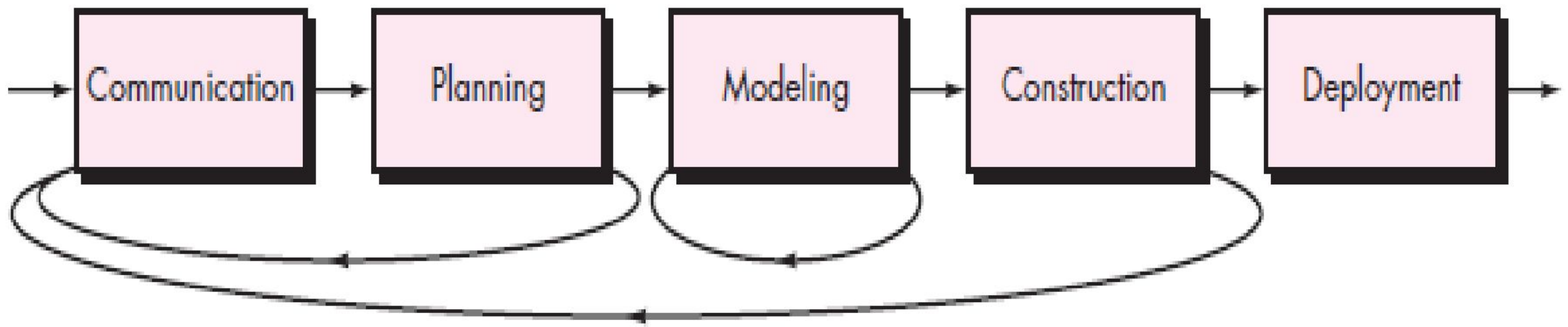
- Executes each of the five framework activities in sequence, beginning with communication and culminating with deployment



(a) Linear process flow

Iterative Process Flow

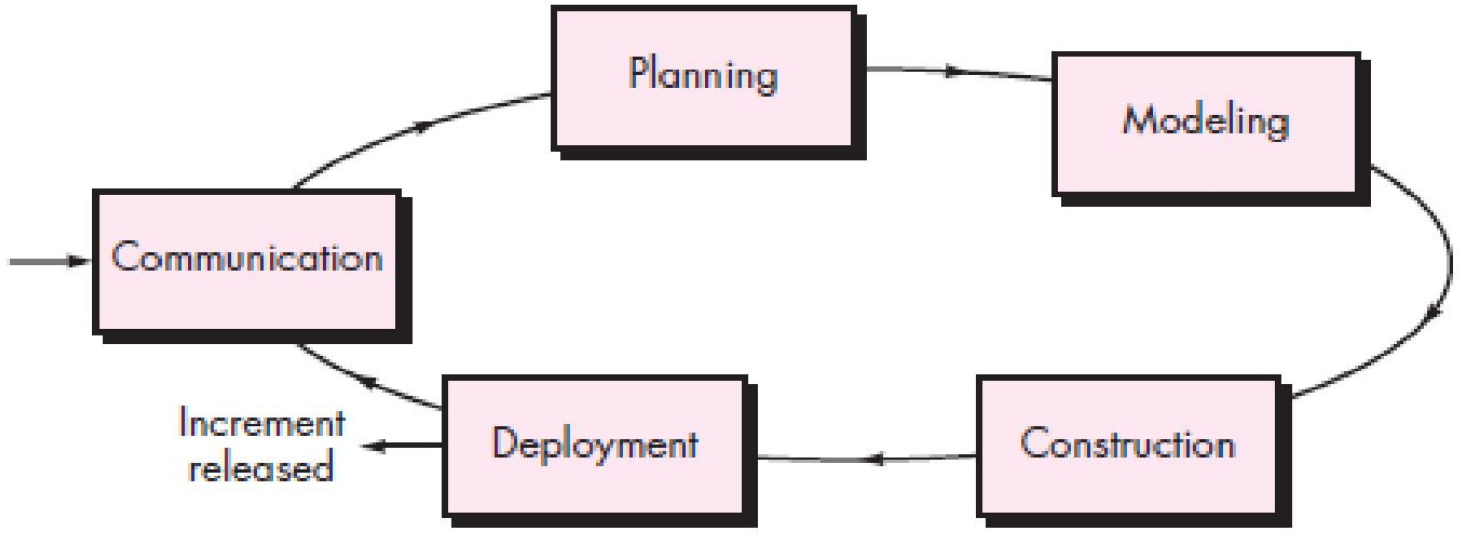
- Repeats one or more of the activities before proceeding to the next



(b) Iterative process flow

Evolutionary Process Flow

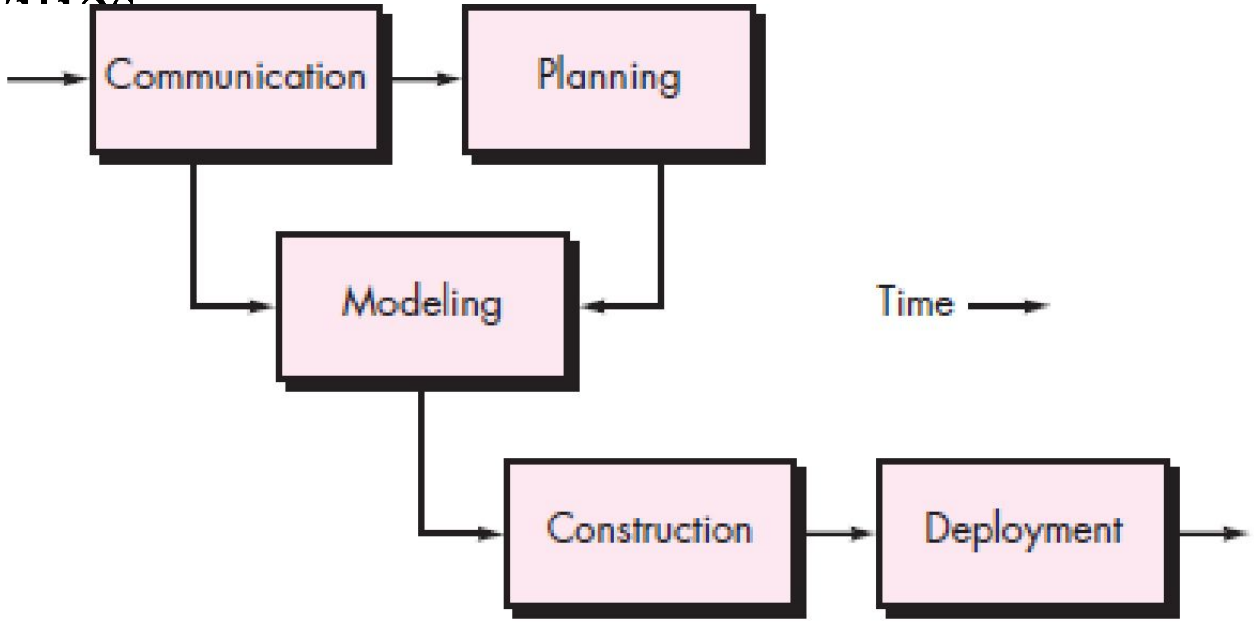
- Executes the activities in a “circular” manner. Each circuit through the five activities leads to a more complete version of the software



(c) Evolutionary process flow

Parallel Process Flow

- Executes one or more activities in parallel with other activities



(d) Parallel process flow

CMMI

- CMMI (Capability Maturity Model Integration) is a proven industry framework to improve product quality and development efficiency for both hardware and software
  - Sponsored by US Department of Defence in cooperation with Carnegie Mellon University and the Software Engineering Institute (SEI)
  - Many companies have been involved in CMMI definition such as Motorola and Ericsson
  - CMMI has been established as a model to improve business results

CMMI

- Vastly improved version of the CMM
- Emphasis on business needs, integration and institutionalization
- CMMI, staged, uses 5 levels to describe the maturity of the organization, same as predecessor CMM

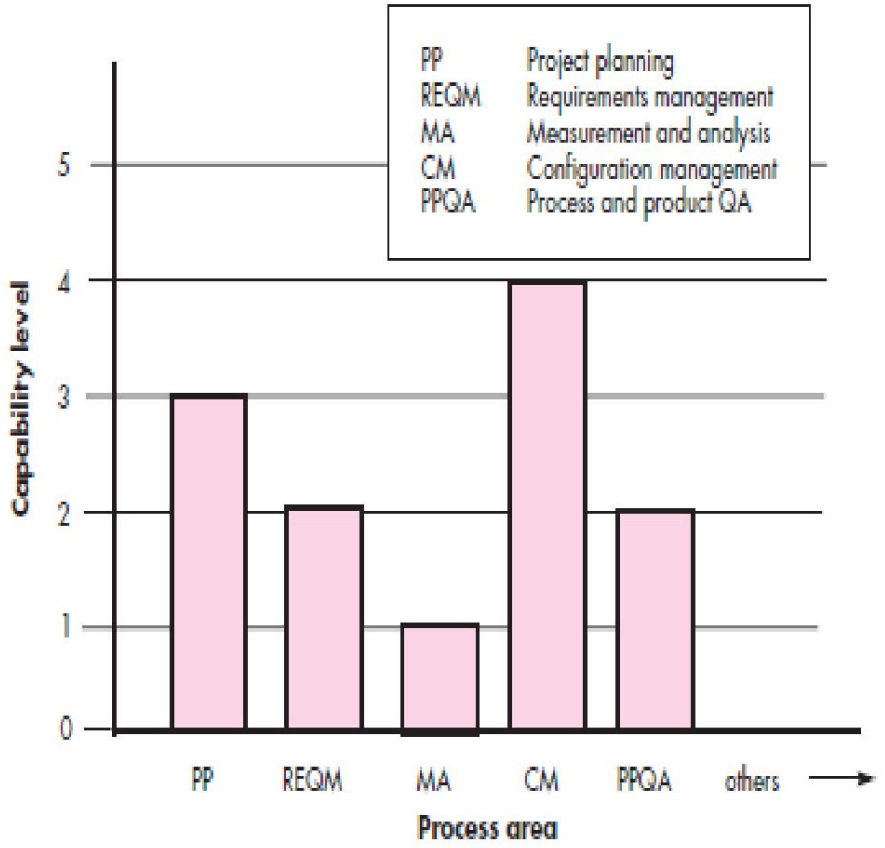
CMMI

The CMMI represents a process meta-model in two different ways:

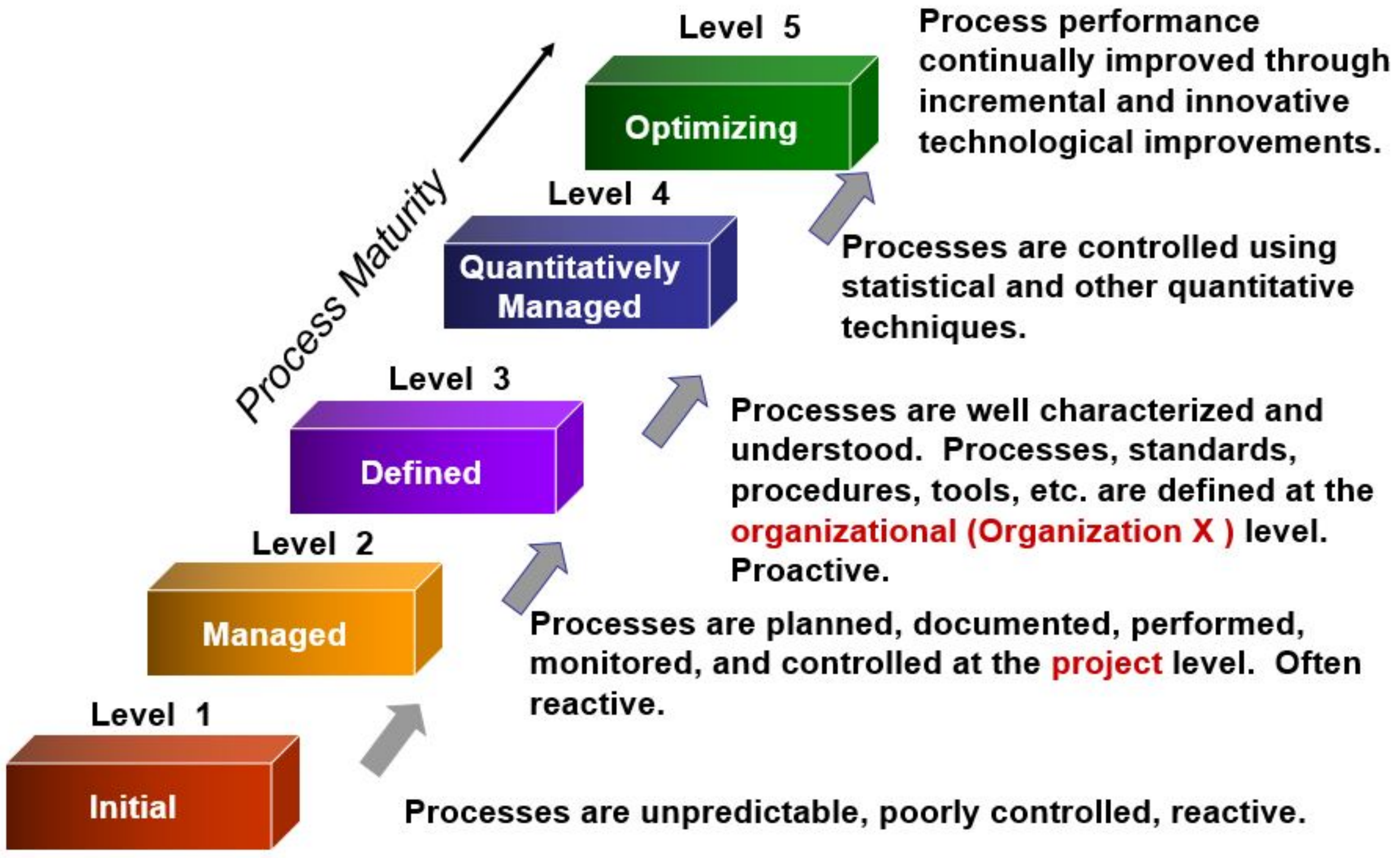
(1) as a "continuous" model and

(2) as a "staged" model.

The continuous CMMI metamodel describes a process in two dimensions as illustrated in Figure



CMMI Maturity Levels



Process performance continually improved through incremental and innovative technological improvements.

Processes are controlled using statistical and other quantitative techniques.

Processes are well characterized and understood. Processes, standards, procedures, tools, etc. are defined at the organizational (Organization X) level. Proactive.

Processes are planned, documented, performed, monitored, and controlled at the project level. Often reactive.

Processes are unpredictable, poorly controlled, reactive.

Principal Model Components of CMMI

1. A set of process areas that are related to software process activities.
2. A number of goals, which are abstract descriptions of a desirable state that should be attained by an organization.
3. A set of good practices, which are descriptions of ways of achieving a goal.

Principal Model Components of CMMI

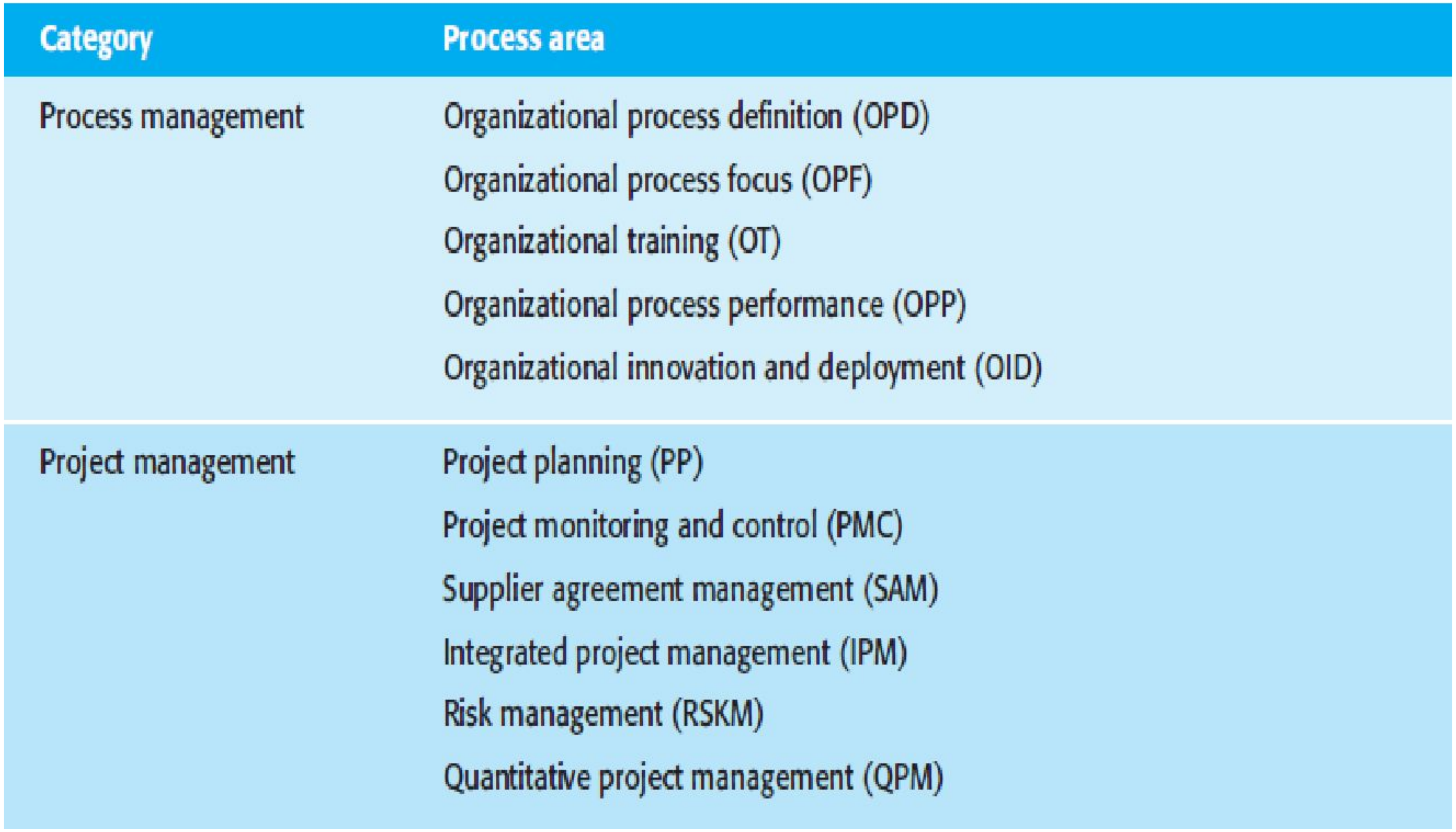
Process Areas:

The CMMI identifies 22 process areas that are relevant to software process capability and improvement.

These are organized into four groups in the continuous CMMI model.

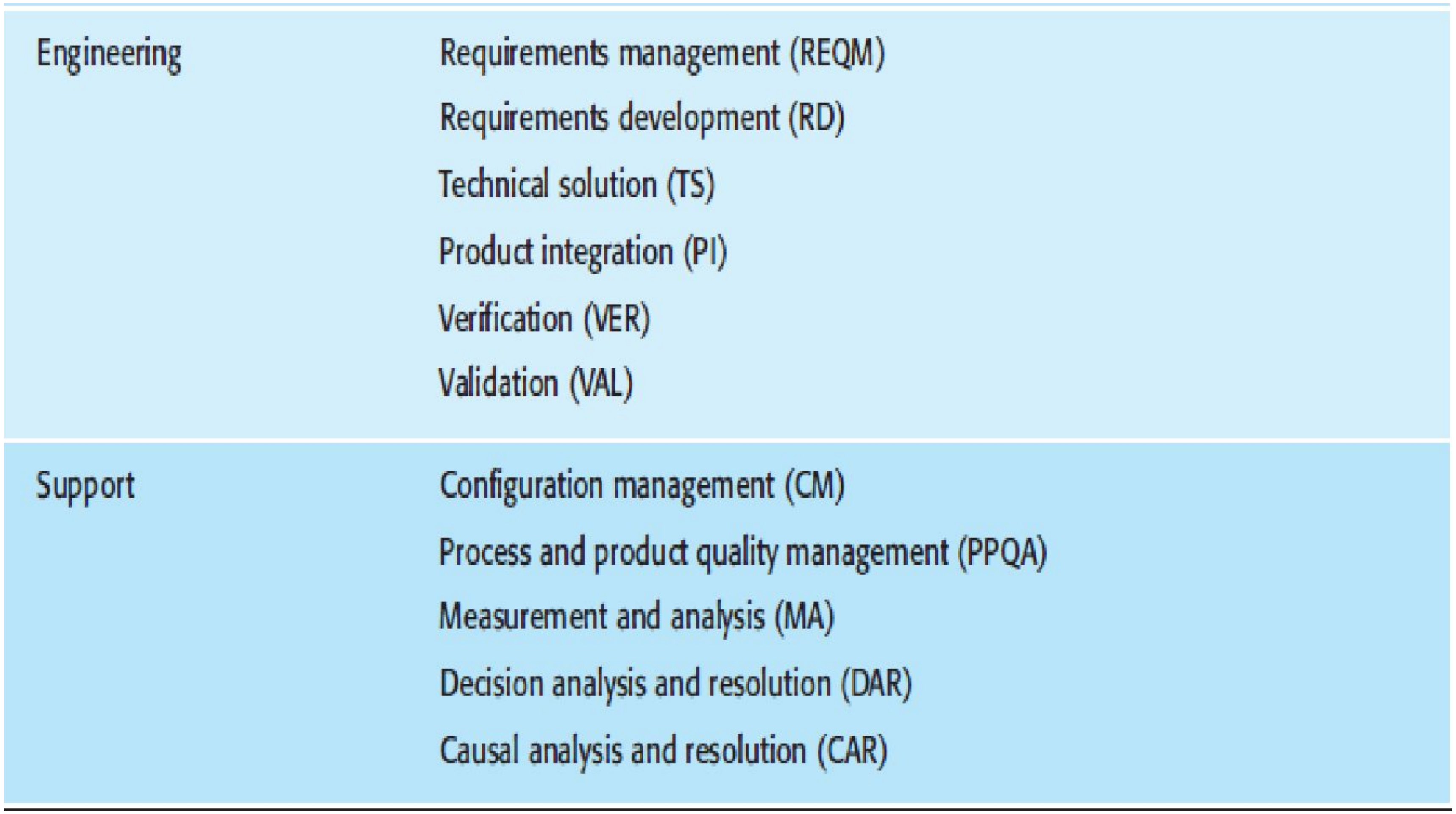
These groups and related process areas are listed in Figure

Principal Model Components of CMMI- Process Areas



Category	Process area
Process management	Organizational process definition (OPD)
	Organizational process focus (OPF)
	Organizational training (OT)
	Organizational process performance (OPP)
	Organizational innovation and deployment (OID)
Project management	Project planning (PP)
	Project monitoring and control (PMC)
	Supplier agreement management (SAM)
	Integrated project management (IPM)
	Risk management (RSKM)
	Quantitative project management (QPM)

Principal Model Components of CMMI- Process Areas



Engineering	Requirements management (REQM)
	Requirements development (RD)
	Technical solution (TS)
	Product integration (PI)
	Verification (VER)
	Validation (VAL)
Support	Configuration management (CM)
	Process and product quality management (PPQA)
	Measurement and analysis (MA)
	Decision analysis and resolution (DAR)
	Causal analysis and resolution (CAR)

Principal Model Components of CMMI- Number of Goals

1. A number of goals, which are abstract descriptions of a desirable state that should be attained by an organization.

Goal	Process area
Corrective actions are managed to closure when the project's performance or results deviate significantly from the plan.	Project monitoring and control (specific goal)
Actual performance and progress of the project are monitored against the project plan.	Project monitoring and control (specific goal)
The requirements are analyzed and validated, and a definition of the required functionality is developed.	Requirements development (specific goal)
Root causes of defects and other problems are systematically determined.	Causal analysis and resolution (specific goal)
The process is institutionalized as a defined process.	Generic goal

Goal	Associated practices
The requirements are analyzed and validated, and a definition of the required functionality is developed.	Analyze derived requirements systematically to ensure that they are necessary and sufficient.
	Validate requirements to ensure that the resulting product will perform as intended in the user's environment, using multiple techniques as appropriate.
Root causes of defects and other problems are systematically determined.	Select the critical defects and other problems for analysis.
	Perform causal analysis of selected defects and other problems and propose actions to address them.
The process is institutionalized as a defined process.	Establish and maintain an organizational policy for planning and performing the requirements development process.
	Assign responsibility and authority for performing the process, developing the work products, and providing the services of the requirements development process.