SOFTWARE ENGINEERING

Presenting by:
B.Pranalini

Index

Unit 1 Part 1:

INTRODUCTION TO SOFTWARE ENGINEERING:

- Software
- The Nature of Software
- Software Engineering
- The Software Process
- Software Engineering practice
- Software Myths
- A Generic Process Model
- Process Assessment and Improvement
- Product and Process
- CMMI

Software

What is Software?

The product that software professionals build and then support over the long term.
Software encompasses:

1) instructions (computer programs) that when executed provide desired features, function, and performance;
2) data structures that enable the programs to adequately store and manipulate information and
3) documentation that describes the operation and use of the programs.

Software products

- Generic products
  - Stand-alone systems that are marketed and sold to any customer who wishes to buy them.
  - Examples – PC software such as editing, graphics programs, project management tools; CAD software; software for specific markets such as appointments systems for dentists.
- Customized products
  - Software that is commissioned by a specific customer to meet their own needs.
  - Examples – embedded control systems, air traffic control software, traffic monitoring systems.

Why Software is Important?

- The economies of ALL developed nations are dependent on software.
- More and more systems are software controlled (transportation, medical, telecommunications, military, industrial, entertainment, etc)
- Software engineering is concerned with theories, methods and tools for professional software development.
- Expenditure on software represents a significant fraction of Gross National Product (GNP) in all developed countries.

Software costs

- Software costs often dominate computer system costs. The costs of software on a PC are often greater than the hardware cost.
- Software costs more to maintain than it does to develop. For systems with a long life, maintenance costs may be several times development costs.
- Software engineering is concerned with cost-effective software development.

Software characteristics

- To gain an understanding of software, it is important to gain characteristics that make it different from other things human being build.

Features of such logical system:

- Software is developed or engineered, it is not manufactured in the classical sense which has quality problem.
- Software doesn't "wear out." but it deteriorates (due to change). Hardware has bathtub curve of failure rate (high failure rate in the beginning, then drop to steady

FIGURE 1.1
Failure curve for hardware

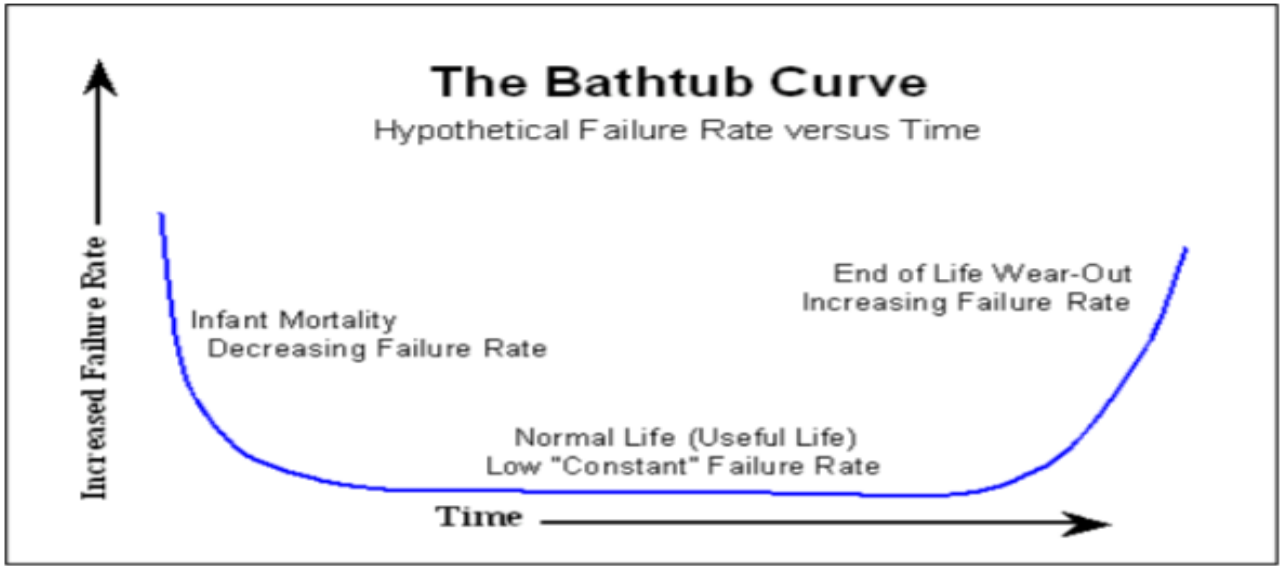
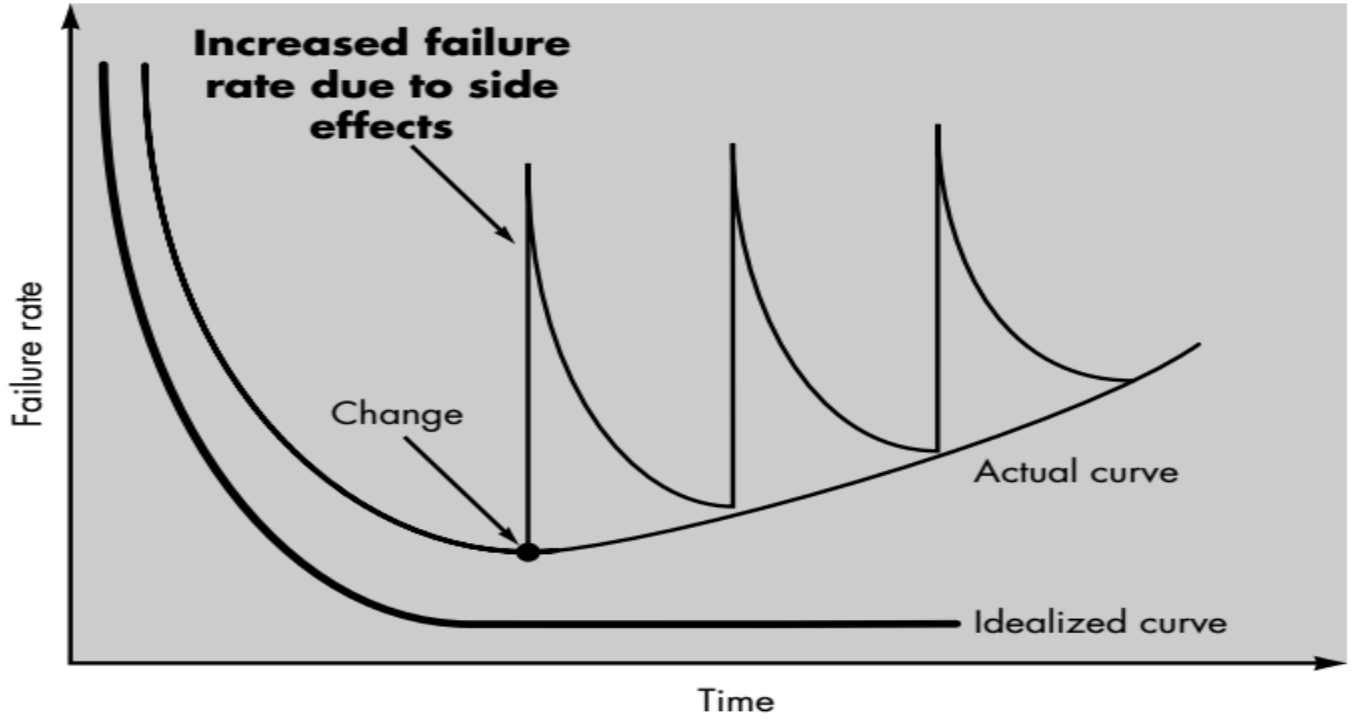


FIGURE 1.2
Idealized and actual failure curves for software



KEY POINT
Software engineering

- Although the industry is moving toward component-based construction (e.g. standard screws and off-the-shelf integrated circuits), most software continues to be custom-built.

  Modern reusable components encapsulate data and processing into software parts to be reused by different programs. E.g. graphical user interface, window, pull-down menus in library etc.

THE CHANGING NATURE OF SOFTWARE

1. System software: such as compilers, editors, file management utilities.
2. Application software: stand-alone programs for specific needs.
3. Engineering/scientific software: Characterized by “number crunching” algorithms. Such as automotive stress analysis, molecular biology, orbital dynamics etc
4. Embedded software resides within a product or system. (key pad control of a microwave oven, digital function of dashboard display in a car)
5. Product-line software focus on a limited marketplace to address mass consumer market. (word processing, graphics, database management)
6. WebApps (Web applications) network centric software. As web 2.0 emerges, more sophisticated computing environments is supported integrated with remote database and business applications.

Software—New Categories

- Open world computing—pervasive, ubiquitous, distributed computing due to wireless networking. How to allow mobile devices, personal computer, enterprise system to communicate across vast network.
- Netsourcing—the Web as a computing engine. How to architect simple and sophisticated applications to target end-users worldwide.
- Open source—”free” source code open to the computing community (a blessing, but also a potential curse!)

Software Engineering Definition

The seminal definition:

[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.

The IEEE definition:

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).

Importance of Software Engineering

- More and more, individuals and society rely on advanced
  software systems. We need to be able to produce reliable and trustworthy systems economically and quickly.
- It is usually cheaper, in the long run, to use software engineering methods and techniques for software systems rather than just write the programs as if it was a personal programming project. For most types of system, the majority of costs are the costs of changing the software after it has gone into use.

Software Engineering - A Layered Technology

FIGURE 2.1
Software engineering layers

- Any engineering approach must rest on organizational commitment to quality which fosters a continuous process improvement culture.
- Process layer as the foundation defines a framework with activities for effective delivery of software engineering technology. Establish the context where products (model, data, report, and forms) are produced, milestone are established, quality is ensured and change is managed.
- Method provides technical how-to's for building software. It encompasses many tasks including communication, requirement analysis, design modeling, program construction, testing and support.

- A process is a collection of activities, actions and tasks that are performed when some work product is to be created. It is not a rigid prescription for how to build computer software. Rather, it is an adaptable approach that enables the people doing the work to pick and choose the appropriate set of work actions and tasks.
- Purpose of process is to deliver software in a timely manner and with sufficient quality to satisfy those who have sponsored its creation and those who will use it.

A Generic View of Software Engineering

Engineering is the analysis, design, construction, verification, and management of technical (or social) entities.

- The work associated with software engineering can be categorized into three generic phases
- The *definition phase* focuses on *what*
  what information is to be processed, what function and performance are desired, what system behavior can be expected,
- The *development phase* focuses on *how*.
  how data are to be structured, how function is to be implemented within a software architecture,
- The *support phase* focuses on *change*
  changing customer requirements.

  Four types of change:

  Correction. *Corrective maintenance* changes the software to correct defects.

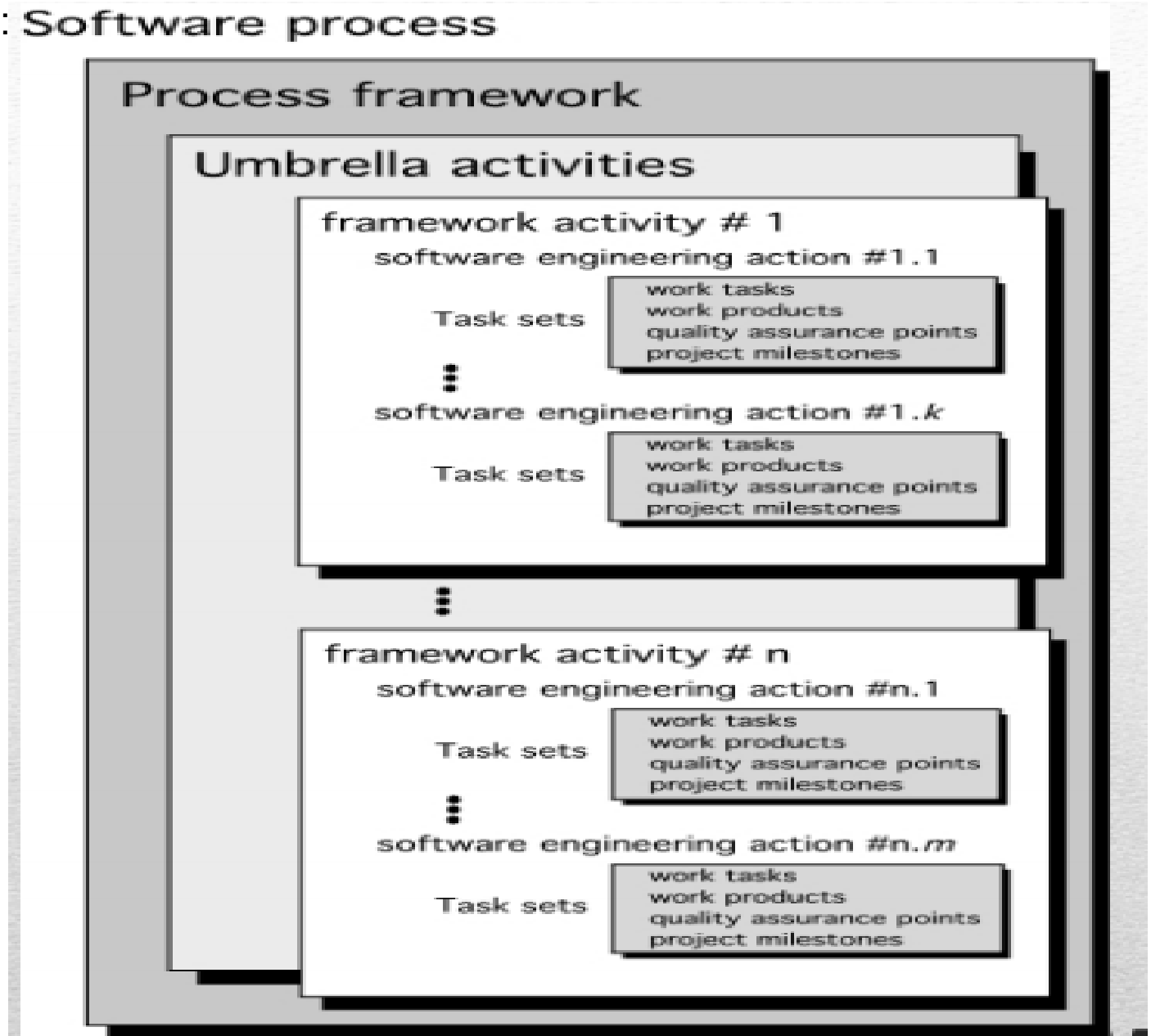
Enhancement. *Perfective maintenance* extends the software beyond its original functional requirements

Prevention. Computer software deteriorates due to change, and because of this, *preventive maintenance,* often called *software reengineering*

- The phases and related steps described in our generic view of software engineering are complemented by a number of *umbrella activities.* Typical activities in this category include:
  • Software project tracking and control
  • Formal technical reviews
  • Software quality assurance
  • Software configuration management
  • Document preparation and production
  • Reusability management
  • Measurement
  • Risk management

Software Process

Fig: Software process



- A software process can be characterized in the above figure.
- A *common process framework* is established by defining a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.
- A number of *task sets*—each a collection of software engineering work tasks, project milestones, work products, and quality assurance points.
- umbrella activities—such as software quality assurance, software configuration management, and measurement—overlay the process model.

In recent years, there has been a significant emphasis on "process maturity."

Five process maturity levels:

Level 1: Initial. The software process is characterized as ad hoc and occasionally even chaotic.
Level 2: Repeatable. Basic project management processes are established to track cost, schedule, and functionality.
Level 3: Defined. The software process for both management and engineering activities is documented, standardized, and integrated into an organizationwide software process.
Level 4: Managed. Detailed measures of the software process and product

Software Engineering Practice

- Practice is collection of Concepts, Principles, Methods & Tools that a software engineer calls upon on a daily basis.
- Practice allows Managers to manage software projects & Software Engineers to build computer programs.

The Essence of SE Practice

1. Understand the Problem – Communication & Analysis
2. Plan a Solution – Modeling & Software Design
3. Carry out the Plan – Code generation
4. Examine the Results – Testing & QA

Core Principles:

To establish mind-set for solid software engineering practice (David Hooker 96).

1. The Reason It All Exists: provide values to users
2. KISS (Keep It Simple, Stupid! As simple as possible)
3. Maintain the Vision (otherwise, incompatible design)
4. What You Produce, Others Will Consume (code with concern for those that must maintain and extend the system)
5. Be Open to the Future (never design yourself into a corner as specification and hardware changes)
6. Plan Ahead for Reuse
7. Think! Place clear complete thought before action produces better results

SOFTWARE MYTHS

- Myths are widely held but false beliefs and views which propagate misinformation and confusion.
- Three types of myth are associated with software:
  - Management myth
  - Customer myth
  - Practitioner's myth

  Management myths. Managers with software responsibility, are often under pressure to maintain budgets, keep schedules from slipping, and improve quality. a software manager often grasps at belief in a software myth, if that belief will lessen the pressure.

  Myth(1): We already have a book that's full of standards and procedures for building software, won't that provide my people with everything they need to know?

  Reality: The book of standards may very well exist, but is it used? Are software practitioners aware of its existence? Answer is "no"

  Myth(2):Each organization feel that they have state-of-art software development tools since they have latest computer.

Myth(3):Adding more programmers when the work is behind schedule can catch up.

Reality: Software development is not a mechanistic process like manufacturing

Myth(4):Outsourcing the software project to third party, we can relax and let that party build it.

Reality: If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsources software projects.

Customer myths. A customer who requests computer software may be a person at the next desk, a technical group down the hall, the marketing/sales department. In many cases, the customer believes myths about software because software managers and practitioners do little to correct misinformation.

Myth(1):General statement of objective is enough to begin writing programs, the details can be filled in later.

Reality: A poor up-front definition is the major cause of failed software efforts.

Myth(2):Software is easy to change because software is flexible.
Reality: It is true that software requirements change, but the impact of change varies with the time at which it is introduced.

Practitioner's myths. Myths that are still believed by software practitioners have been fostered by 50 years of programming culture. During the early days of software, programming was viewed as an art form. Old ways and attitudes die hard.

Myth(1):Once the program is written, the job has been done.

Reality: Someone once said that "the sooner you begin 'writing code', the longer it'll take you to get done."
Myth(2):Until the program is running, there is no way of assessing the quality.

Reality: One of the most effective software quality assurance mechanisms can be applied from the inception of a project—the *formal technical review.*
Myth(3):The only deliverable work product is the working program

Reality: A working program is only one part of a *software configuration* that includes many elements.
Myth(4):Software Engineering creates voluminous and unnecessary documentation and invariably slows down software development.
Reality: Software engineering is not about creating documents. It is

A Generic Process Model

A generic process framework for software engineering defines five framework activities:

- Communication: communicate with customer to understand objectives and gather requirements.
- Planning: creates a “map” defines the work by describing the tasks, risks and resources, work products and work schedule.
- Modeling: Create a “sketch”, what it looks like architecturally, how the constituent parts fit together and other characteristics.
- Construction: code generation and the testing.
- Deployment: Delivered to the customer who evaluates the products and provides feedback based on the evaluation.

These five framework activities can be used to all software development regardless of the application domain, size of the project

For many software projects, these framework activities are applied iteratively as a project progresses. Each iteration produces a software increment that provides a subset of overall software features and functionality.

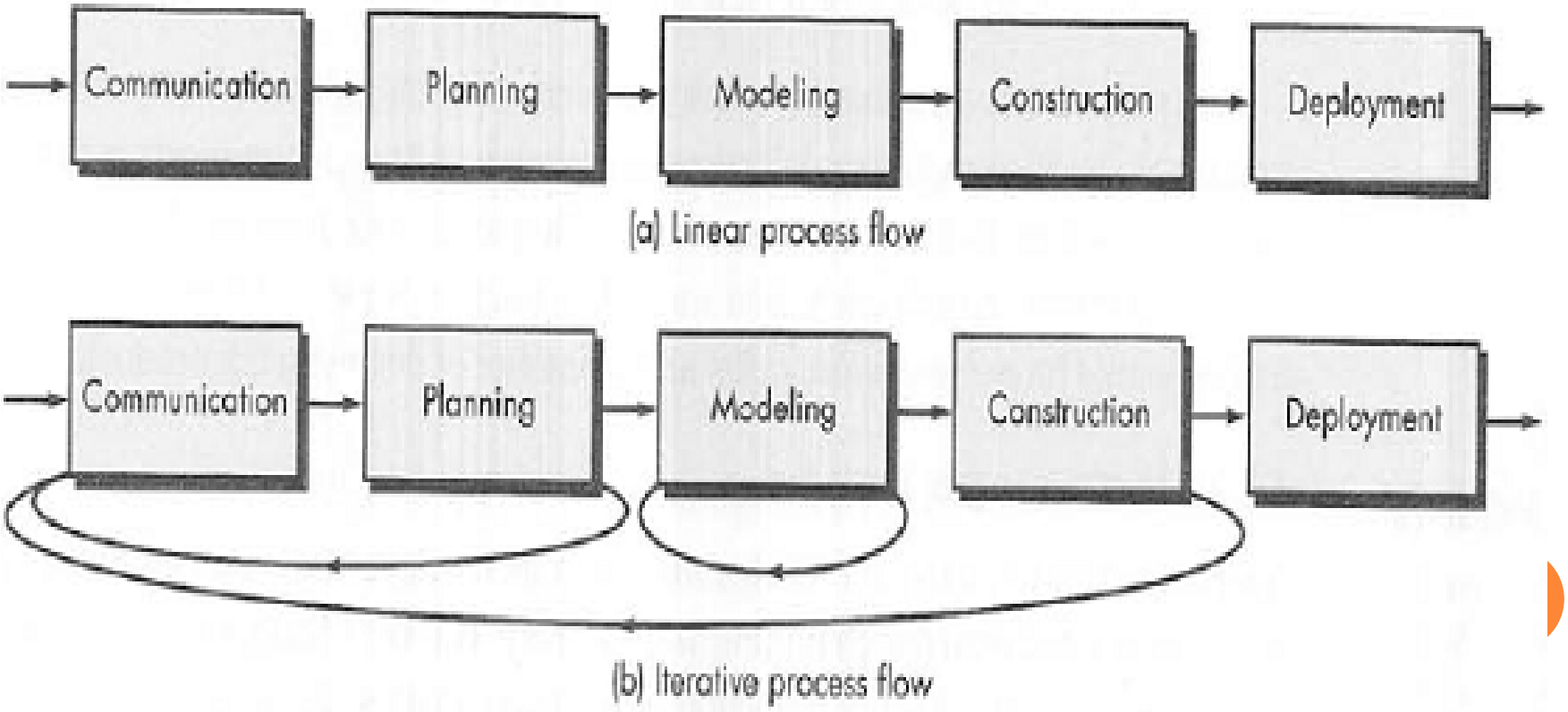
In addition, a set of umbrella activities-

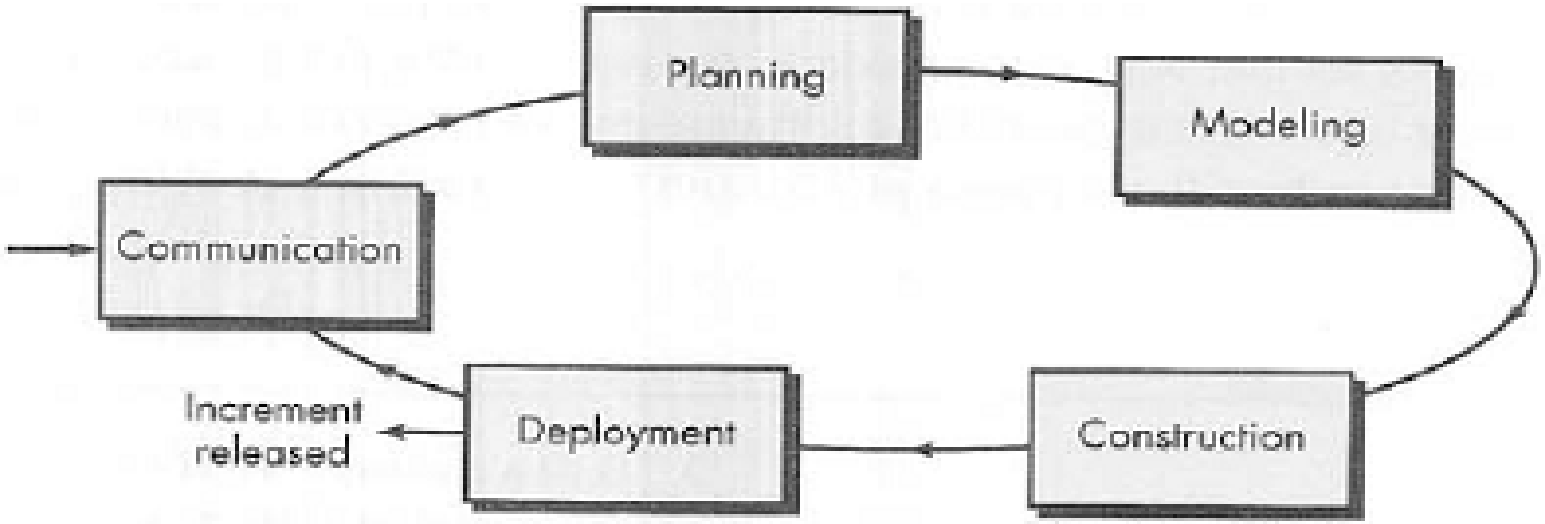
The five process framework activities helps a team to manage and control, progress, quality, change, and risk.

- Software project tracking and control: assess progress against the plan and take actions to maintain the schedule.
- Risk management: assesses risks that may affect the outcome and quality.
- Software quality assurance: defines and conduct activities to ensure quality.
- Technical reviews: assesses work products to uncover and remove errors before going to the next activity.
- Measurement: define and collects process, project, and product measures to ensure stakeholder's needs are met.
- Software configuration management: manage the effects of change throughout the software process.

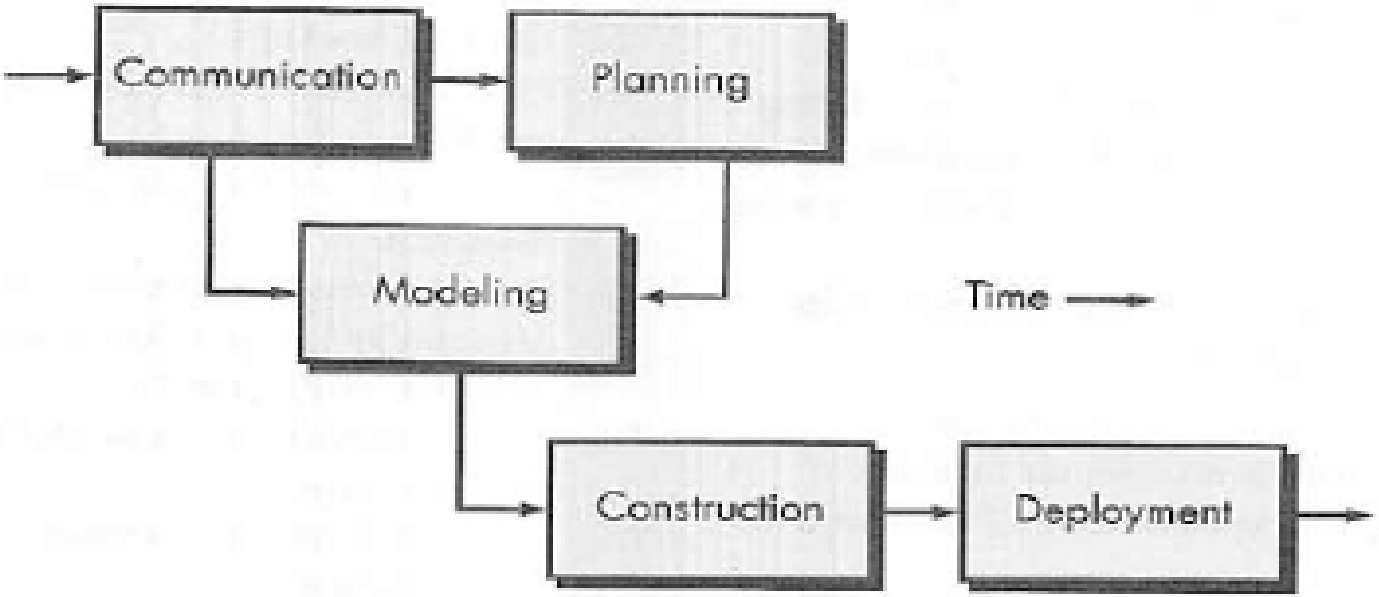
- Reusability management: defines criteria for work product reuse and establishes mechanism to achieve reusable components.
- Work product preparation and production: create work products such as models, documents, logs, forms and lists.

- The framework activities and the actions and tasks





(c) Evolutionary process flow



(d) Parallel process flow

- Linear process flow executes each of the five activities in sequence.
- An iterative process flow repeats one or more of the activities before proceeding to the next.
- An evolutionary process flow executes the activities in a circular manner. Each circuit leads to a more complete version of the software.
- A parallel process flow executes one or more activities in parallel with other activities (modeling for one aspect of the software in parallel with construction of another aspect of the software).

Identifying a Task Set

- Before we can proceed with the process model, a key question: what actions are appropriate for a framework activity given the nature of the problem, the characteristics of the people and the stakeholders?
- A task set defines the actual work to be done to accomplish the objectives of a software engineering action.
  - A list of the task to be accomplished
  - A list of the work products to be produced
  - A list of the quality assurance filters to be applied
- For example, a small software project requested by one person with simple requirements, the communication activity might encompass little more than a phone all with the stakeholder. Therefore, the only necessary action is phone conversation, the work tasks of this action are:

  1. Make contact with stakeholder via telephone.

Example of a Task Set for Elicitation

The task sets for Requirements gathering action for a simple project may include:

1. Make a list of stakeholders for the project.
2. Invite all stakeholders to an informal meeting.
3. Ask each stakeholder to make a list of features and functions required.
4. Discuss requirements and build a final list.
5. Prioritize requirements.
6. Note areas of uncertainty.

Example of a Task Set for Elicitation

The task sets for Requirements gathering action for a big project may include:

1. Make a list of stakeholders for the project.
2. Interview each stakeholders separately to determine overall wants and needs.
3. Build a preliminary list of functions and features based on stakeholder input.
4. Schedule a series of facilitated application specification meetings.
5. Conduct meetings.
6. Produce informal user scenarios as part of each meeting.
7. Refine user scenarios based on stakeholder feedback.
8. Build a revised list of stakeholder requirements.
9. Use quality function deployment techniques to prioritize requirements.
10. Package requirements so that they can be delivered incrementally.
11. Note constraints and restrictions that will be placed on the system.

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.
- Stated in more general terms, a process pattern provides you with a template [Amb98]—a consistent method for describing problem solutions within the context of the software process.
  (defined at different levels of abstraction)
  1. Problems and solutions associated with a complete process model (e.g. prototyping).
  2. Problems and solutions associated with a framework activity (e.g. planning) or
  3. an action with a framework activity (e.g. project

Process Pattern Types

- Stage patterns—defines a problem associated with a framework activity for the process. It includes multiple task patterns as well. For example, Establishing Communication would incorporate the task pattern Requirements Gathering and others.
- Task patterns—defines a problem associated with a software engineering action or work task and relevant to successful software engineering practice
- Phase patterns—define the sequence of framework activities that occur with the process, even when the overall flow of activities is iterative in nature. Example includes SprialModel or Prototyping.

An Example of Process Pattern

- Describes an approach that may be applicable when stakeholders have a general idea of what must be done but are unsure of specific software requirements.
- Pattern name. Requirements Unclear
- Intent. This pattern describes an approach for building a model that can be assessed iteratively by stakeholders in an effort to identify or solidify software requirements.
- Type. Phase pattern
- Initial context. Conditions must be met

  (1)stakeholders have been identified;

  (2) a mode of communication between stakeholders and the software team has been established;

  (3) the overriding software problem to be solved has been identified by stakeholders ;

  (4) an initial understanding of project scope, basic business requirements and project constraints has been

- Problem. Requirements are hazy or nonexistent. stakeholders are unsure of what they want.
- Solution. A description of the prototyping process would be presented here.
- Resulting context. A software prototype that identifies basic requirements. (modes of interaction, computational features, processing functions) is approved by stakeholders. Following this,

  1. This prototype may evolve through a series of increments to become the production software or

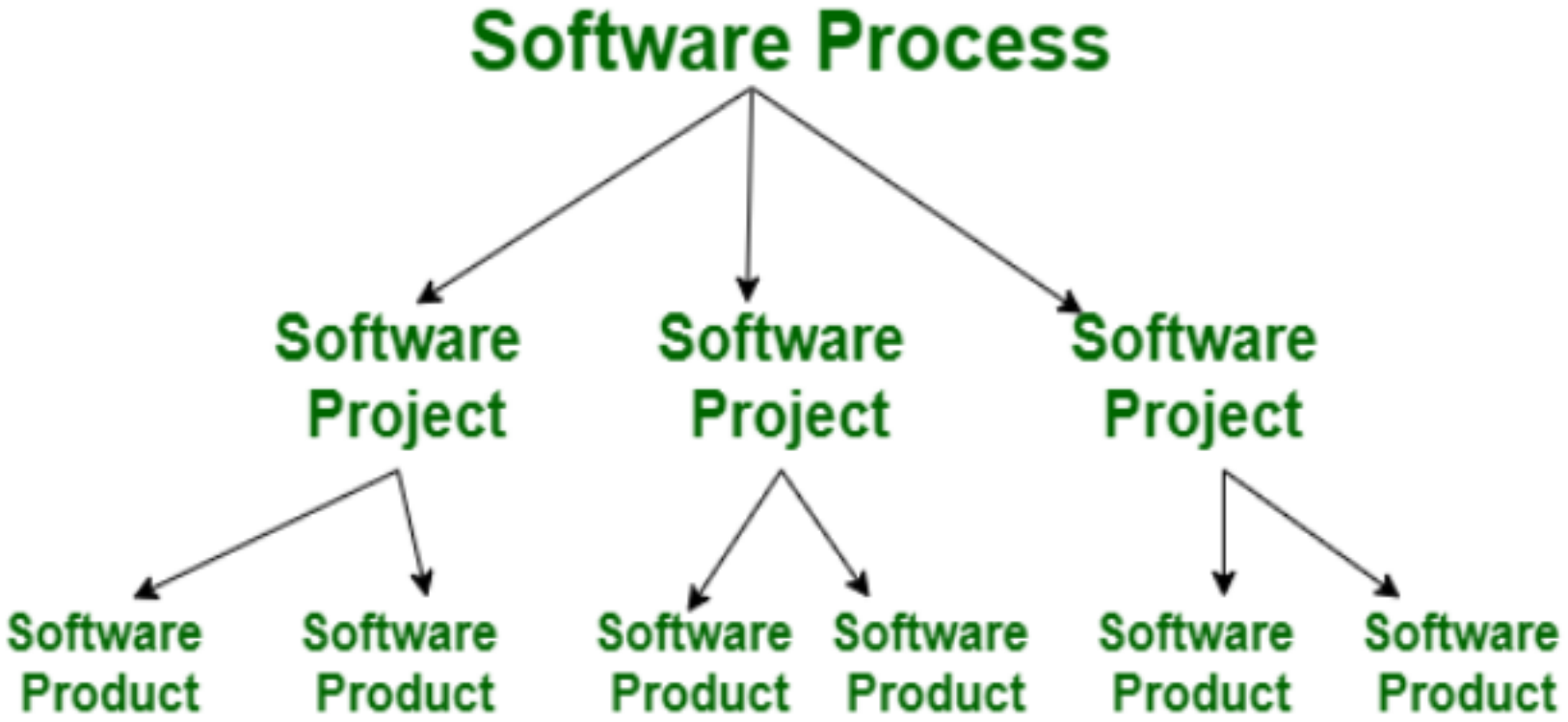
  2. the prototype may be discarded.
- Related patterns. CustomerCommunication, IterativeDesign, IterativeDevelopment, CustomerAssessment, RequirementExtraction.

Process Assessment and Improvement

SP cannot guarantee that software will be delivered on time, meet the needs, or has the desired technical characteristics. However, the process can be assessed to ensure that it meets a set of basic process criteria that have been shown to be essential for a successful software engineering.

- Standard CMMI Assessment Method for Process Improvement (SCAMPI) — provides a five step process assessment model that incorporates five phases: initiating, diagnosing, establishing, acting and learning.
- CMM-Based Appraisal for Internal Process Improvement (CBA IPI)—provides a diagnostic technique for assessing the relative maturity of a software organization; uses the SEI CMM as the basis for the assessment [Dun01]
- SPICE—The SPICE (ISO/IEC15504) standard defines a set of requirements for software process assessment. The intent of the standard is to assist organizations in developing an objective evaluation of the efficacy of any defined software process. [ISO08]
- ISO 9001:2000 for Software—a generic standard that applies to

Product and Process



- **Product:**

  Software engineering methods, tools and techniques for creating a collection of similar software systems from a shared set of software assets using a common means of production.

- **Process:**

  Main functionalities of the software and the constrains around them.

  If the process is weak, the end product will undoubtedly suffer.

  But an obsessive over- reliable on process is also dangerous. In a brief essay, Margaret davis (DAV95) comments on the duality of the product and process.

  About every ten years give or take five, The software community redefines “the problem” by shifting its focus from product issues to process issues. Thus, we have embraced structured programming languages (product) followed by structured analysis methods (process) followed by data encapsulation (product) followed by the current emphasis on the software engineering institutes software development capability maturity model (process) [followed by object-oriented methods, followed by agile software development]

CAPABILITY MATURITY MODEL INTEGRATION (CMMI)

- Developed by SEI(Software Engineering institute)
- "CMMI is a development model created after a study of data collected from organizations that contracted with the U.S. department of defense, who founded the research"
- Assess the process model followed by an organization and rate the organization with different levels.
- A set of software engineering capabilities should be present as organizations reach different levels of process capability and maturity.

CMMI process meta model can be represented in different ways

1.A continuous model
2.A staged model

Continuous model:

-Lets organization select specific improvement that best meet its business objectives and minimize risk

-Levels are called capability levels.

-Describes a process in 2 dimensions

-Each process area is assessed against specific goals and practices and is rated according to the following capability levels.

CMMI

- Six levels of CMMI
  - Level 0:Incomplete
  - Level 1:Performed
  - Level 2:Managed
  - Level 3:Defined
  - Level 4:Quantitatively managed
  - Level 5:Optimized

• Incomplete -Process is adhoc . Objective and goal of process areas are not known
• Performed –Goal, objective, work tasks, work products and other activities of software process are carried out
• Managed -Activities are monitored, reviewed, evaluated and controlled
• Defined -Activities are standardized, integrated and documented
• Quantitatively Managed -Metrics and indicators are available to measure the process and quality
• Optimized - Continuous process improvement based on quantitative feed back from the user
-Use of innovative ideas and techniques, statistical quality control and other methods for process improvement.

<u>Staged model:</u>
- This model is used if you have no clue of how to improve the process for quality software.
- It gives a suggestion of what things other organizations have found helpful to work first

Capability and Maturity Levels of CMMI

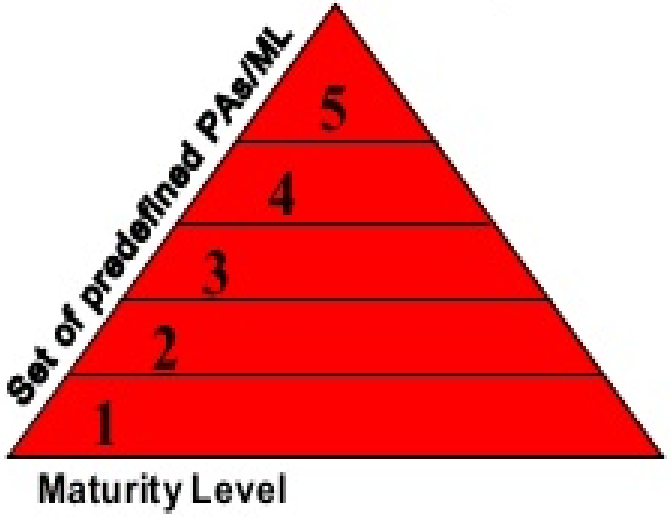
Levels	*Continuous Representation Capability Levels*	*Staged Representation Maturity Levels*
Level 0	Incomplete	N/A
Level 1	Performed	Initial
Level 2	Managed	Managed
Level 3	Defined	Defined
Level 4	Quantitatively Managed	Quantitatively Managed
Level 5	Optimizing	Optimizing

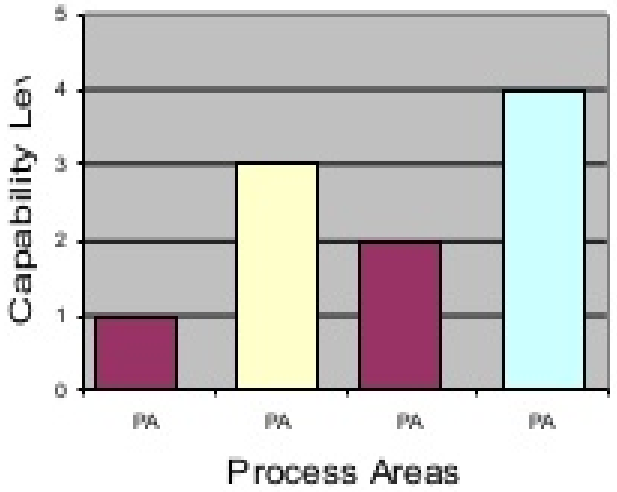
CMMI Model – Two Representations

STAGED

Provides pre-defined roadmap for *organizational improvement*, based on *proven grouping* of processes and associated organizational relationships.

Structured by Maturity Levels.





CONTINUOUS

Provides *flexibility* for organizations to choose *which processes* to emphasize for improvement, as well as *how much* to improve each process.

Structured by Categories of Process Areas.

Thank You