1...

Introduction to System Engineering

Learning Objectives...

- To understand the concept of System.
- To learn about Software in detail.
- To study about Software Engineering.
- To learn the McCall's Quality Factor and Software Process.

1.1 DEFINITION OF SYSTEM

- In software engineering, a "system" refers to a set of interconnected components or elements that work together to achieve a specific set of objectives or functionalities. These components can include hardware, software, data, and human resources. The system's purpose is to perform a particular function or solve a problem, and it operates as a whole to deliver its intended services or outcomes.
- The study of system concepts has three basic implications:
  1. A system must be designed to achieve a predetermined objective.
  2. Interrelationships and interdependence must exist among the components.
  3. The objectives of the organization as a whole have a higher priority than the objectives of its subsystems.

1.1.1 Characteristic of a System

1. **Organization:** It implies structure and order. It is the arrangement of components that helps to achieve objectives.
2. **Interaction:** It refers to the manner in which each component functions with other components of the system.

3. **Interdependence:** It means that parts of the organization or computer system depend on one another. They are coordinated and linked together according to a plan. One subsystem depends on the output of another subsystem for proper functioning.
4. **Integration:** It refers to the holism of systems. It is concerned with how a system is tied together.
5. **Central Objective:** A system should have a central objective. Objectives may be real or stated. Although a stated objective may be the real objective, it is not uncommon for an organization to state one objective and operates to achieve another. The important point is that users must know the central objective of a computer application early in the analysis for a successful design and conversion.

1.1.2 Basic Components

- The functioning units of system mean the basic elements of the system which are interrelated. These are the basic components of the system.
- Every system is made up of a set of interrelated elements or basic components. These components are the various parts of the system.

Table 1.1

Sr. No	System	Components
1.	Education	Student, teacher, library, laboratory, buildings etc.
2.	Computer	Mouse, display unit, ALU, hard disk, keyboard

- All systems having common main three components in which all they are described.
- Those basic components are input, processor and output as shown in fig 1.1.

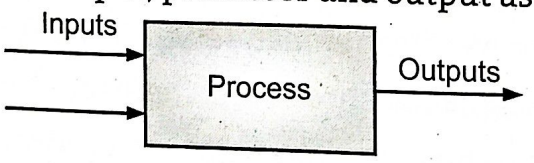

Fig. 1.1

- The basic components of a system typically include:
  1. **Inputs:** This is the information or resources that enter the system. Inputs can be data, materials, energy, or other resources necessary for the system to function.
  2. **Process:** This is the core activity or set of activities that transform inputs into outputs. The process involves operations, methods, or procedures that are used to handle the inputs.
  3. **Outputs:** This is the result or product of the process. Outputs are the end results or outcomes produced by the system, which can be in the form of data, goods, services, or other deliverables.

1.1.3 Elements of the System

- All the characteristics of the system are determined by the system elements, their properties and relationships.
- The basic system elements are input, processor and output (explained in section 1.1.2).
- Universal model of the system is made up of system elements like input, processor and output using basic concept like control and feedback to keep the system balance.
- The universal system model is as shown in Fig. 1.2.

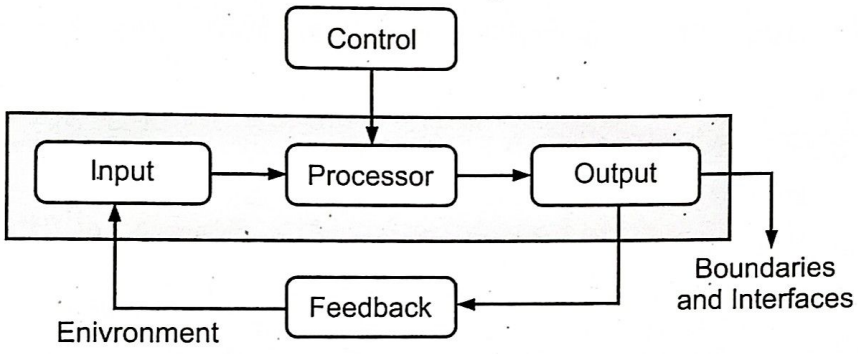

Fig. 1.2: Elements of the system

- **Input:** This is the information or resources that enter the system. Inputs can be data, materials, energy, or other resources necessary for the system to function.
- **Process:** This is the core activity or set of activities that transform inputs into outputs. The process involves operations, methods, or procedures that are used to handle the inputs.
- **Output:** This is the result or product of the process. Outputs are the end results or outcomes produced by the system, which can be in the form of data, goods, services, or other deliverables.
- **Feedback:** This component provides information about the output or performance of the system. Feedback is used to make adjustments or improvements to ensure the system meets its goals effectively.
- **Control:** This involves the mechanisms or rules used to regulate and direct the process to achieve desired outcomes. Control ensures that the system operates within specified parameters and makes adjustments based on feedback.
- **Environment:** The environment encompasses external factors that can affect the system. It includes everything outside the system that interacts with or influences its performance, such as market conditions, regulations, or physical surroundings.
- **Boundaries:** These define the limits of the system, distinguishing what is inside the system from what is outside. Boundaries help in identifying what components are parts of the system and which are external factors.

  Interface means interaction of different system parts with each other as interaction of the system with the system outside the boundaries.

- **Subsystems:** These are smaller, individual systems within a larger system. Each subsystem has its own components and functions, contributing to the overall operation of the larger system.

1.1.4 Types of System

1. **Open System:**
   - Interacts with its environment.
   - Exchanges information, energy, or material with the outside world.
   - **Example:** A university system interacts with students, staff, government, etc.
2. **Closed System:**
   - Does not interact with its environment.
   - Self-contained and isolated.
   - **Example:** A chemical reaction in a sealed container (ideal example; almost no true closed systems exist in practice).
3. **Physical System:**
   - Tangible and made of physical components.
   - Can be touched or seen.
   - **Example:** A computer, a power plant, a car engine.
4. **Abstract System:**
   - Conceptual or non-physical.
   - Made of ideas, models, or logical relationships.
   - **Example:** A software algorithm, mathematical models, business processes.
5. **Deterministic System:**
   - The outcome is predictable and follows a predefined path or rules.
   - No randomness involved.
   - **Example:** A calculator, 2 + 2 always equals 4.
6. **Probabilistic System:**
   - The outcome involves uncertainty or randomness.
   - Based on probability, predictions are not guaranteed.
   - **Example:** Weather forecasting system, stock market analysis system.
7. **Information System:**
   - A system designed to collect, process, store, and distribute information.
   - Supports decision-making, coordination, control, analysis, and visualization.
   - **Components:** Hardware, software, data, procedures, and people.
   - **Example:** Library management system, banking information system.

8. **Management Information System (MIS):**
   - A type of information system specifically focused on managing organizational operations.
   - Provides timely and relevant information to managers for decision-making.
   - Converts data from TPS (Transaction Processing Systems) into useful reports and summaries.
   - **Functions:** Planning, controlling, and managing organizational operations.
   - **Example:** Sales reporting system, employee performance dashboard.

Table 1.2: Summary Table

System Type	Description	Example
Open System	Interacts with environment.	Online shopping system.
Closed System	No interaction with environment (theoretical).	Sealed automation system.
Physical System	Tangible, made of real components.	Computer, server room.
Abstract System	Conceptual, logical, or theoretical.	Database schema, flowchart.
Deterministic System	Predictable output given input.	Calculator.
Probabilistic System	Output involves randomness or probability.	AI model, stock market analysis.
Information System	Handles data collection, storage, and distribution.	School information system.
MIS	Supports management and decision-making using processed information.	Business performance dashboard.

1.1.5 System Components

1. **Hardware:**
   - The physical infrastructure on which the software runs.
   - Includes servers, networking devices, user devices (PCs, phones), etc.
   - **Example:** A web server hosting an application.
2. **Software**
   - The actual codebase and applications.
   - Includes both **system software** (like operating systems) and **application software** (user-facing programs).

- **Layers of Software:**
  - **Application Layer:** The actual program or app users interact with.
  - **Middleware:** Software that connects different apps or services (e.g., APIs, messaging queues).
  - **Operating System:** Manages hardware and basic system operations.

3. **Data:**
   - Information processed and stored by the system.
   - Can be structured (databases), semi-structured (JSON, XML), or unstructured (documents, images).
   - **Components:**
     - Databases
     - Data Warehouses
     - File Systems
4. **Procedures (Processes):**
   - The operations and workflows that define how tasks are carried out.
   - Includes business logic, algorithms, and operational protocols.
   - **Example:** Login verification process, order fulfillment workflow.
5. **People (Users and Stakeholders):**
   - Anyone who interacts with or is impacted by the system.
   - Roles include:
     - End-users (consumers, employees)
     - Developers
     - System administrators
     - Business analysts
     - Project managers
6. **Network/Communication:**
   - Infrastructure and protocols used to connect system components, especially in distributed systems.
   - **Examples:** APIs, Internet/intranet, HTTP, TCP/IP protocols
7. **Interface**
   - How users or systems interact with the software.
   - **Types:**
     - **User Interface (UI):** Graphical (GUI), command-line (CLI)
     - **System Interface:** APIs, services, or SDKs
8. **Security:**
   - Mechanisms to protect the system from threats and unauthorized access.

- Includes:
  - Authentication and authorization
  - Encryption
  - Firewalls, intrusion detection
  - Data integrity checks

9. **Integration Modules:**
   - Components responsible for connecting with external systems or third-party services.
   - **Examples:**
     - Payment gateways
     - External APIs (e.g., Google Maps, Stripe)

1.2 DEFINITION OF SOFTWARE

- Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be a collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called software product. Engineering on the other hand, is all about developing products, using well defined, scientific principles and methods.

1.2.1 Characteristics of Software

1. **Software Manufactured or Engineered:**
   - Software is developed or engineered; it is not manufactured in the classical sense.
   - In both software development and hardware manufacturing activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are nonexistent for software. Both activities require the construction of a "product", but the approaches are different. Software costs are concentrated in engineering. This means that software projects cannot be managed as if they were manufacturing projects.
2. **Software doesn't "wear out"**

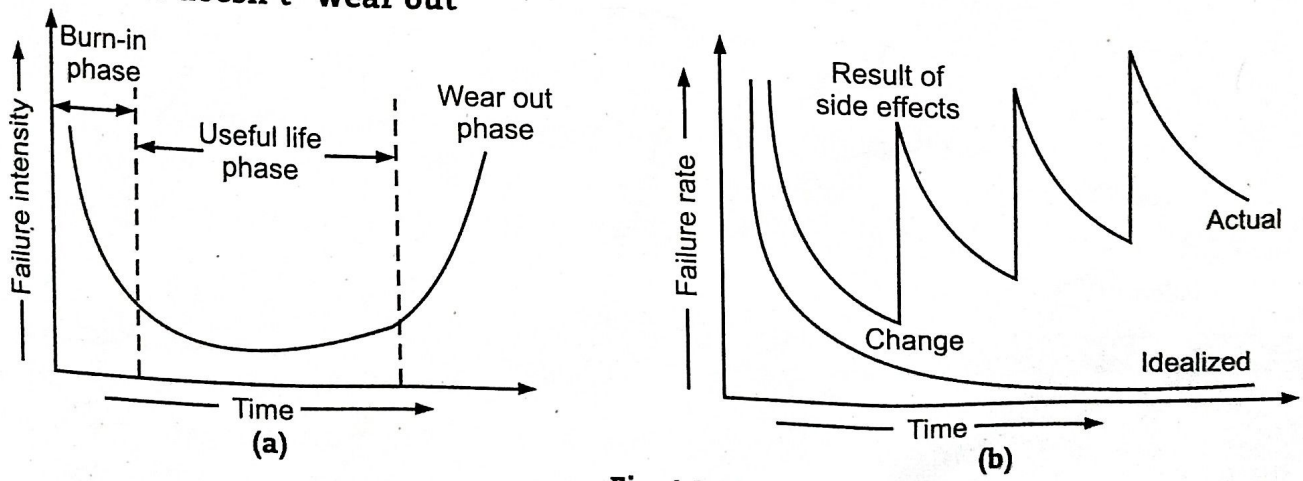

Fig. 1.3

- Fig. 1.3 (a) depicts failure rate as a function of time for hardware. The relationship often called the "bathtub curve", indicates that hardware exhibits relatively high failure rates early in its life due to manufacturing defects; defects are corrected and the failure rate drops to a steady-state level for some period of time. As time passes, however, the failure rate rises again as hardware components suffer from the cumulative effects of dust, vibration, temperature extremes, etc. Stated simply, the hardware begins to wear out. Software is not susceptible to the environmental maladies that cause hardware to wear out. In theory, the failure rate curve for software should take the form of the "idealized curve" shown in Fig. 1.3 (b). Undiscovered defects will cause high failure rates early in the life of a program. However, these are corrected and the curve flattens as shown. Software doesn't wear out. But it does deteriorate! During its life, software will undergo change. As changes are made, it is likely that errors will be introduced, causing the failure rate curve to spike as shown in the "actual curve". Before the curve can return to the original steady-state failure rate, another change is requested, causing the curve to spike again. Slowly, the minimum failure rate level begins to rise–the software is deteriorating due to change.

3. **Most Software is Custom Built:**
   - Although the industry is moving toward component-based construction, most software continues to be custom built.
   - Standard screws and off-the-shelf integrated circuits are only two of thousands of standard components that are used by mechanical and electrical engineers as they design new systems. In the hardware world, component reuse is a natural part of the engineering process. In the software world, it is something that has only begun to be achieved on a broad scale. A software component should be designed and implemented so that it can be reused in many different programs. Modern reusable components encapsulate both data and the processing that is applied to the data, enabling the software engineer to create new applications from reusable parts.
   - For example, today's interactive user interfaces are built with reusable components that enable the creation of graphics windows, pull-down menus, and a wide variety of interaction mechanisms. The data structures and processing detail required to build the interface are contained within a library of reusable components for interface construction.

1.3 DEFINITION OF SOFTWARE ENGINEERING

- Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

1.3.1 Layered Technology of Software Engineering

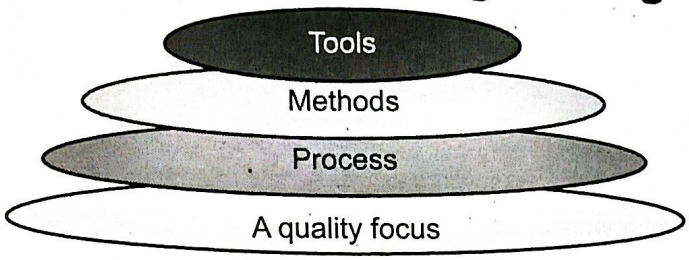

Fig. 1.4: Software engineering layers

- **Quality focus:** Bedrock that supports Software Engineering.
- **Process:** Foundation for software Engineering.
- **Methods:** Provide technical for how to building software.
- **Tools:** Provide semi-automatic and automatic support to methods.

1.3.2 Need for software Engineering

- The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.
- Following are some of the needs stated:
  - **Large software:** It is easier to build a wall than a house or building, likewise, as the size of the software becomes large, engineering has to step to give it a scientific process.
  - **Scalability:** If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.
  - **Cost:** As hardware industry has shown its skills and huge manufacturing has lower down the price of computer and electronic hardware. But, cost of the software remains high if proper process is not adapted.
  - **Dynamic Nature:** Always growing and adapting nature of the software hugely depends upon the environment in which the user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where the software engineering plays a good role.
  - **Quality Management:** Better process of software development provides better and quality software product.

1.4 McCALL'S QUALITY FACTORS

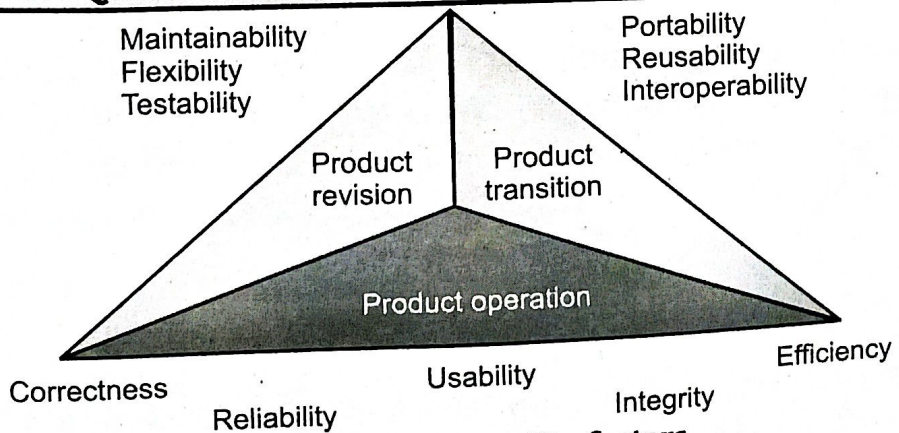

Fig. 1.5: McCall's quality factors

- This model classifies all software requirements into 11 software quality factors. The 11 factors are grouped into three categories – product operation, product revision, and product transition factors.
  - **Product operation factors:** Correctness, Reliability, Efficiency, Integrity, Usability.
  - **Product revision factors:** Maintainability, Flexibility, Testability.
  - **Product transition factors:** Portability, Reusability, Interoperability.

1.4.1 Product Operation Software Quality Factors

- According to McCall's model, product operation category includes five software quality factors, which deal with the requirements that directly affect the daily operation of the software. They are as follows:

1. **Correctness:**
   - These requirements deal with the correctness of the output of the software system.
   - The required accuracy of output that can be negatively affected by inaccurate data or inaccurate calculations.
   - The completeness of the output information, which can be affected by incomplete data.
   - The up-to-dateness of the information defined as the time between the event and the response by the software system.
   - The availability of the information.
   - The standards for coding and documenting the software system.
2. **Reliability:**
   - Reliability requirements deal with service failure. They determine the maximum allowed failure rate of the software system, and can refer to the entire system or to one or more of its separate functions.
3. **Efficiency:**
   - It deals with the hardware resources needed to perform the different functions of the software system.
   - It includes processing capabilities (given in MHz), its storage capacity (given in MB or GB) and the data communication capability (given in MBPS or GBPS).
   - It also deals with the time between recharging of the system's portable units, such as, information system units located in portable computers, or meteorological units placed outdoors.
4. **Integrity:**
   - This factor deals with the software system security, that is, to prevent access to unauthorized persons, also to distinguish between the group of people to be given read as well as write permit.

5. **Usability:**
   - Usability requirements deal with the staff resources needed to train a new employee and to operate the software system.

1.4.2 Product Revision Quality Factors

- According to McCall's model, three software quality factors are included in the product revision category. These factors are as follows:

1. **Maintainability:**
   - This factor considers the efforts that will be needed by users and maintenance personnel to identify the reasons for software failures, to correct the failures, and to verify the success of the corrections.
2. **Flexibility:**
   - This factor deals with the capabilities and efforts required to support adaptive maintenance activities of the software. These include adapting the current software to additional circumstances and customers without changing the software. This factor's requirements also support perfective maintenance activities, such as changes and additions to the software in order to improve its service and to adapt it to changes in the firm's technical or commercial environment.
3. **Testability:**
   - Testability requirements deal with the testing of the software system as well as with its operation. It includes predefined intermediate results, log files, and also the automatic diagnostics performed by the software system prior to starting the system, to find out whether all components of the system are in working order and to obtain a report about the detected faults. Another type of these requirements deals with automatic diagnostic checks applied by the maintenance technicians to detect the causes of software failures.

1.4.3 Product Transition Software Quality Factor

- According to McCall's model, three software quality factors are included in the product transition category that deals with the adaptation of software to other environments and its interaction with other software systems. These factors are as follows:

1. **Portability**
   - Portability requirements tend to the adaptation of a software system to other environments consisting of different hardware, different operating systems, and so forth. The software should be possible to continue using the same basic software in diverse situations.

2. **Reusability:**
   - This factor deals with the use of software modules originally designed for one project in a new software project currently being developed. They may also enable future projects to make use of a given module or a group of modules of the currently developed software. The reuse of software is expected to save development resources, shorten the development period, and provide higher quality modules.
3. **Interoperability:**
   - Interoperability requirements focus on creating interfaces with other software systems or with other equipment firmware. For example, the firmware of the production machinery and testing equipment interfaces with the production control software.

1.5 THE SOFTWARE PROCESS

1.5.1 Software Process Model

- Software Processes is a coherent set of activities for specifying, designing, implementing and testing software systems. A software process model is an abstract representation of a process that presents a description of a process from some particular perspective. There are many different software processes but all involve:
  - **Specification:** Defining what the system should do.
  - **Design and implementation:** Defining the organization of the system and implementing the system.
  - **Validation:** Checking that it does what the customer wants.
  - **Evolution:** Changing the system in response to changing customer needs.

Types of Software Process Model:

- Software processes, methodologies and frameworks range from specific prescriptive steps that can be used directly by an organization in day-to-day work, to flexible frameworks that an organization uses to generate a custom set of steps tailored to the needs of a specific project or group. In some cases a "sponsor" or "maintenance" organization distributes an official set of documents that describe the process.

Software Process and Software Development Lifecycle Model

- One of the basic notions of the software development process is SDLC models which stand for Software Development Life Cycle models. There are many development life cycle models that have been developed in order to achieve different required objectives.

- The models specify the various stages of the process and the order in which they are carried out. The most used, popular and important SDLC models are given below:
  - Waterfall model
  - V model
  - Incremental model
  - RAD model
  - Agile model
  - Iterative model
  - Spiral model
  - Prototype model

1.5.2 Software Process Framework Activities

Process Framework

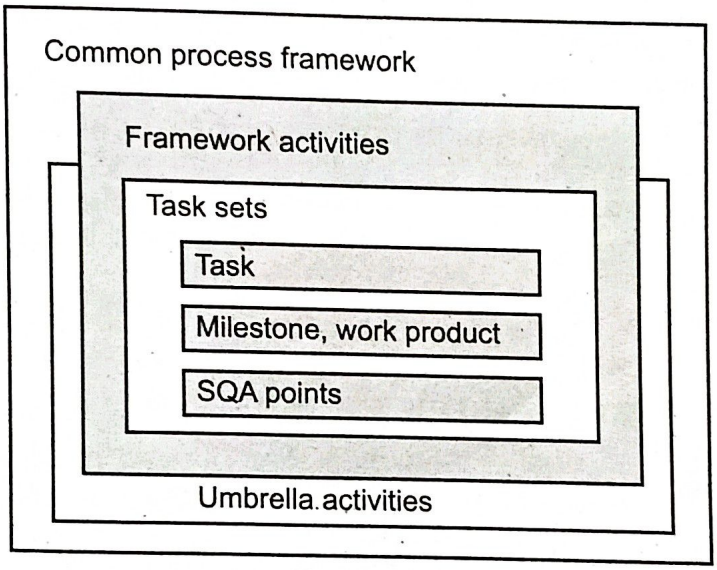

Fig. 1.6: A software process framework

- A software process is a framework for the activities, actions, and tasks that are required to build high-quality software. Each framework activity is populated by a set of software engineering actions. Each software engineering action is defined by a task set that identifies the work tasks that are to be completed, the work products that will be produced, the quality assurance points that will be required, and the milestones that will be used to indicate progress.
- 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 the process. 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. A linear process flow executes each of the five framework activities in sequence, beginning with communication and culminating with deployment.

- 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 through the five activities leads to a more complete version of the software. A parallel process flow executes one or more activities in parallel with other activities.

Software Process Framework Activities:

- The Software process framework is required for representing common process activities. Five framework activities are described in a process framework for software engineering. Communication, planning, modeling, construction, and deployment are all examples of framework activities. Each engineering action defined by a framework activity comprises a list of needed work outputs, project milestones, and Software Quality Assurance (SQA) points.
- Used as a basis for the description of process models Generic process activities. Let's explain each:

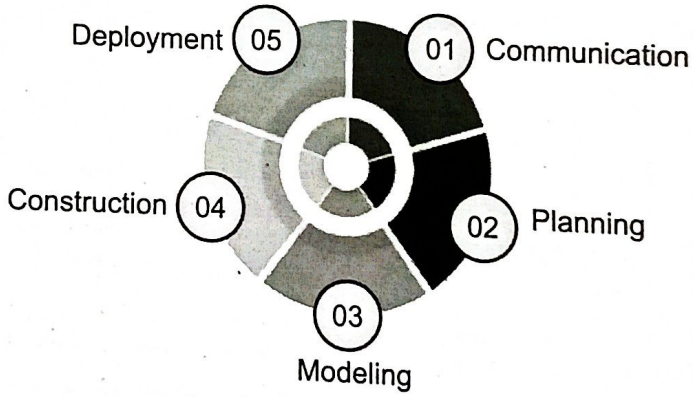

Fig. 1.7: Software Process Framework Activities

- Communication
- Planning
- Modeling
- Construction
- Deployment

1. **Communication:**

- **Definition:** Communication involves gathering requirements from customers and stakeholders to determine the system's objectives and the software's requirements.
- **Activities:**
  - **Requirement Gathering:** Engaging with consumers and stakeholders through meetings, interviews, and surveys to understand their needs and expectations.

- **Objective Setting:** Clearly defining what the system should achieve based on the gathered requirements.

2. **Planning:**

- **Definition:** Planning involves establishing an engineering work plan, describing technical risks, listing resource requirements, and defining a work schedule.
- **Activities:**
  - **Work Plan:** Creating a detailed plan that outlines the tasks and activities needed to develop the software.
  - **Risk Assessment:** Identifying potential technical risks and planning how to mitigate them.
  - **Resource Allocation:** Determining the resources (time, personnel, tools) required for the project.
  - **Schedule Definition:** Setting a timeline for completing different phases of the project.

3. **Modeling:**

- **Definition:** Modeling involves creating architectural models and designs to better understand the problem and work towards the best solution.
- **Activities:**
  - **Analysis of Requirements:** Breaking down the gathered requirements to understand what the system needs to do.
  - **Design:** Creating architectural and detailed designs that outline how the software will be structured and how it will function.

4. **Construction:**

- **Definition:** Construction involves creating code, testing the system, fixing bugs, and confirming that all criteria are met.
- **Activities:**
  - **Code Generation:** Writing the actual code based on the design models.
  - **Testing:** Running tests to ensure the software works as intended, identifying and fixing bugs.

5. **Deployment:**

- **Definition:** Deployment involves presenting the completed or partially completed product to customers for evaluation and feedback, then making necessary modifications based on their input.
- **Activities:**
  - **Product Release:** Delivering the software to users, either as a full release or in stages.

- **Feedback Collection:** Gathering feedback from users about their experience with the software.
- **Product Improvement:** Making changes and improvements based on user feedback to enhance the product.

1.5.3 Umbrella Activities

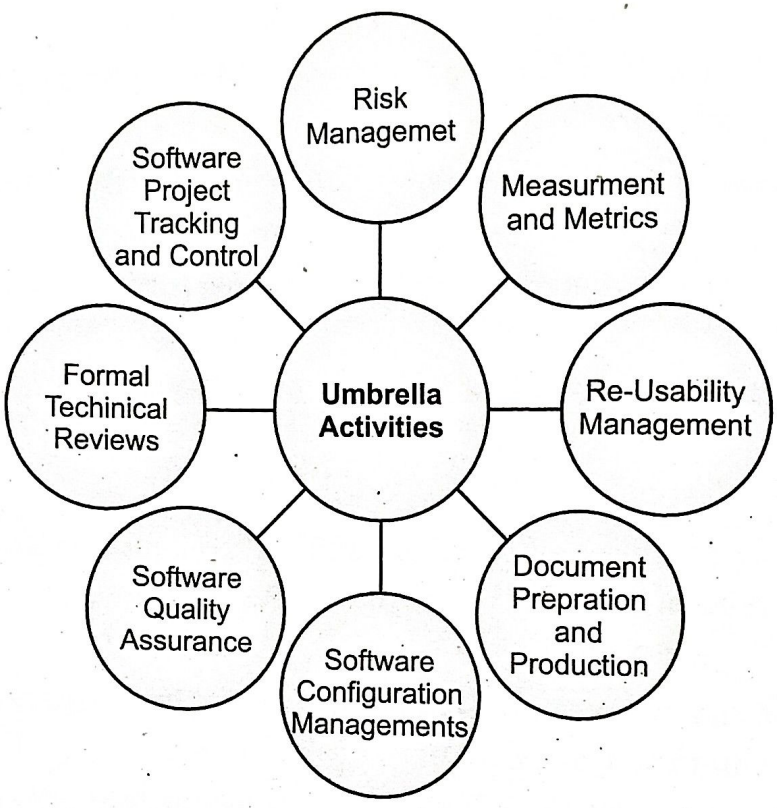

Fig. 1.8: Umbrella activities

- Umbrella Activities that take place during a software development process for improved project management and tracking.

1. **Software Project Tracking and Control:** This is an activity in which the team can assess progress and take corrective action to maintain the schedule. Take action to keep the project on time by comparing the project's progress against the plan.
2. **Risk Management:** The risks that may affect project outcomes or quality can be analyzed. Analyze potential risks that may have an impact on the software product's quality and outcome.
3. **Software Quality Assurance:** These are activities required to maintain software quality. Perform actions to ensure the product's quality.
4. **Formal Technical Reviews:** It is required to assess engineering work products to uncover and remove errors before they propagate to the next activity. At each level of the process, errors are evaluated and fixed.
5. **Software Configuration Management:** Managing the configuration process when any change in the software occurs.

6. **Work product Preparation and Production:** The activities to create models, documents, logs, forms, and lists are carried out.
7. **Reusability Management:** It defines criteria for work product reuse. Reusable work items should be backed up, and reusable software components should be achieved.
8. **Measurement:** In this activity, the process can be defined and collected. Also, project and product measures are used to assist the software team in delivering the required software.

Check Your Understanding

1. A system can be defined as:
   (a) A set of unrelated components
   (b) A group of interrelated components working toward a common goal
   (c) A random collection of processes
   (d) A mechanical device
2. Which of the following is not a characteristic of a system?
   (a) Organization (b) Interaction
   (c) Independence (d) Integration
3. What are the basic components of a system?
   (a) Input, Process, Output, Feedback, Control
   (b) Input, Control, Budget, Goals
   (c) Database, Feedback, Hardware
   (d) Input, Software, Output
4. Which is not considered an element of a system?
   (a) Input (b) Process (c) Noise (d) Environment
5. Which of the following is a closed system?
   (a) Human body (b) Ecosystem
   (c) Economy (d) Washing machine
6. Control, feedback, and environment are part of:
   (a) Programming languages (b) Operating system
   (c) System components (d) None of the above

7. Software is defined as:
   (a) The physical part of a computer
   (b) The non-physical, coded instructions executed by a computer
   (c) The hardware driver only
   (d) The memory and CPU
8. Which of the following is true about software?
   (a) Software wears out over time
   (b) Software is a manufactured product
   (c) Software can be repaired physically
   (d) Software never requires maintenance
9. Most software is:
   (a) Manufactured using machines
   (b) Produced on assembly lines
   (c) Custom-built to specific needs
   (d) Recyclable
10. Software Engineering is:
   (a) Engineering only mechanical systems
   (b) Designing and manufacturing computer hardware
   (c) Systematic development and maintenance of software
   (d) Use of software by end users
11. Software Engineering is needed because:
   (a) Software is cheap to build
   (b) It helps in solving large, complex problems systematically
   (c) Hardware is more important
   (d) Software needs no planning
12. Which of the following is a Product Operation factor?
   (a) Maintainability (b) Flexibility (c) Reliability (d) Portability
13. Which of these is under Product Revision?
   (a) Maintainability (b) Usability (c) Portability (d) Efficiency
14. Product Transition includes:
   (a) Testability (b) Integrity (c) Portability (d) Correctness

15. A software process model is:
    (a) A hardware design plan
    (b) A framework for developing software
    (c) A license agreement
    (d) A user manual
16. Which is not a software process framework activity?
    (a) Communication (b) Modeling (c) Construction (d) Packaging
17. Umbrella activities include:
    (a) Risk management (b) Coding
    (c) Testing (d) Implementation

Answers

1. (b)	2. (c)	3. (a)	4. (c)	5. (d)	6. (c)	7. (b)	8. (b)	9. (c)	10. (c)
11. (b)	12. (c)	13. (a)	14. (c)	15. (b)	16. (d)	17. (a)			

Practice Questions

Q.I Answer the following questions in short:

1. What is a system?
2. List the main characteristics of a system.
3. What are the basic components of a system?
4. Classify systems with examples (open/closed, deterministic/non-deterministic)
5. Define software.
6. Define software engineering.
7. What are McCall's quality factors?
8. What is a software process?

Q.II Answer the following questions in detail:

1. What is a system? Explain with examples.
2. Define a system and explain its importance in software engineering.
3. Explain the main characteristics of a system.
4. Write a short note on basic components of a system.
5. Explain the role of each component in a system.
6. Explain open and closed systems.
7. Explain different types of systems with their characteristics.

8. Differentiate between software and hardware.
9. Explain the characteristics of software in detail.
10. Discuss why software does not wear out.
11. Why is most software custom built?
12. Explain what it means that software is manufactured or engineered.
13. Explain the goals and significance of software engineering.
14. Describe the layered technology model of software engineering.
15. Explain each layer and its role in software development.
16. Why is software engineering necessary?
17. Explain different software process models (Waterfall, Spiral, Agile).
18. Describe the activities involved in the software process framework.
19. What are umbrella activities in the software process? Give examples and their importance.

***