Chp 1

1) What is software? what are the fundamental types of software?

In a computer system, the software is basically a set of instructions or commands that tells a computer what to do. Or in other words, the software is a computer program that provides a set of instructions to execute a user's commands and tell the computer what to do. For example, like MS-Word, MS-Excel, PowerPoint, etc. There are two fundamental types of software:

1) Generic Software Products: These are the stand - alone system that is produced by the development organization and sold on open market to any customer who can buy them. Example – databases, word processors.

2) Customized Software Products: These are the systems which are commissioned by customer. Example – Systems support a particular business process, Air traffic control systems

Generic software products	Custom software products
The generic software development is done for developing a general purpose software	Customer software development is done to develop a software product as per the needs of customer.
In this development process, the software developers must depict the end-user's specifications.	is development process, the end user requirements can be aggregated by communicating by them.
From designing and marketing perspective, this type of development is very difficult.	This development does not require marketing, because it is developed for appropriate group of users.
Large number of users may be using this kind of software.	This type of software is used by limited number of users.
Quality of the product is not a preference for generic software.	Quality is the main criterion in customer software product. Best quality of the product is focused for customer or company.
Development team controls the process of generic software development.	Customer determines the process of software development in this type of product.
Generally, the software developed is economical. There may be some hidden costs such as installation and implementation cost	Software product is of high cost as the product for customer is developed.
Example of generic software product development is Word editing software.	Inventory control and management system are examples of customer software

2) List and explain the various attributes of the software

The characteristics of any software product include features which are displayed by the product when it is installed and put in use. They are not the services which are provided by the product. Instead, they have related to the products dynamic behaviour and the use made of the product.

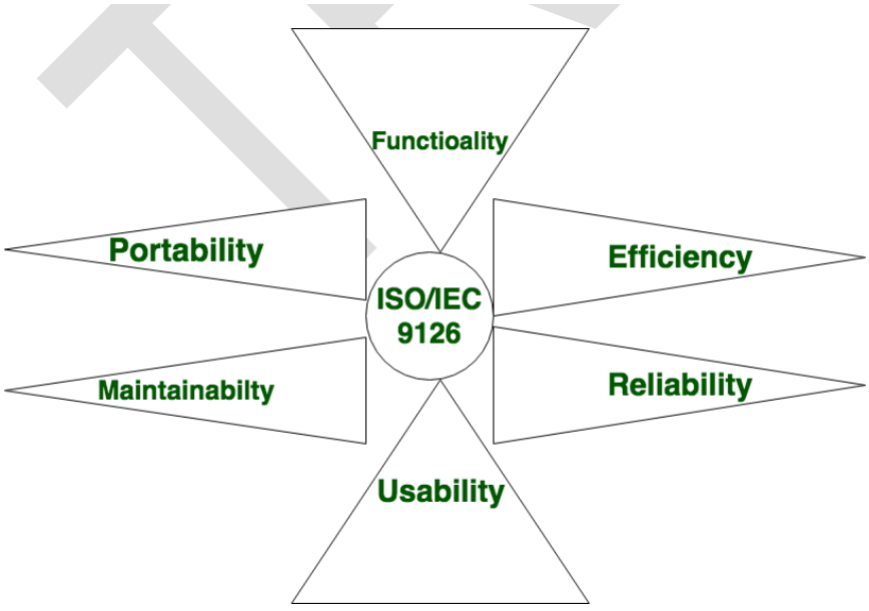
Examples of these attributes are:

Efficiency, reliability, robustness, maintainability, etc. However, the relative importance of these characteristics varies from one software system to another

Product Characteristics	Description
Maintainability	The software should evolve to meet the changing demands of the clients.
Dependability	Dependability includes various characteristics. Dependable software should never cause any physical or economic damage at the time of system failure
Efficiency	The software application should overuse system resources like memory and processor cycle.
Usability	The software application should have specific UI and documentation

Characteristics of good Software

Software is defined as a collection of computer programs, procedures, rules, and data. Software Characteristics are classified into six major components:



These components are described below:

Functionality: It refers to the degree of performance of the software against its intended purpose.

Reliability: A set of attributes that bears on the capability of software to maintain its level of performance under the given condition for a stated period.

Efficiency: It refers to the ability of the software to use system resources in the most effective and efficient manner. The software should make effective use of storage space and executive command as per desired timing requirements.

Usability: It refers to the extent to which the software can be used with ease. the amount of effort or time required to learn how to use the software.

Maintainability: It refers to the ease with which the modifications can be made in a software system to extend its functionality, improve its performance, or correct errors.

Portability: A set of attributes that bears on the ability of software to be transferred from one environment to another, without or minimum changes

3) What is software engineering? Explain in detail

Software engineering is defined as a process of analysing user requirements and then designing, building, and testing software application which will satisfy those requirements. IEEE, in its standard 610.12-1990, defines software engineering as the application of a systematic, disciplined, which is a computable approach for the development, operation, and maintenance of software

Software Crisis & its Solution

What was the Software Crisis?

- It was in the late 1960s when many software projects failed.
- Many software became over budget. Output was an unreliable software which is expensive to maintain.
- Larger software was difficult and quite expensive to maintain.
- Lots of software not able to satisfy the growing requirements of the customer.
- Complexities of software projects increased whenever its hardware capability increased
- Demand for new software increased faster compared with the ability to generate new software.

All the above issues lead to ‘Software Crisis.’

The Solution

Solution was to the problem was transforming unorganized coding effort into a software engineering discipline. These engineering models helped companies to streamline operations and deliver software meeting customer requirements.

- The late 1970s saw the widespread uses of software engineering principles.

- In the 1980s saw the automation of software engineering process and growth of (CASE) Computer-Aided Software Engineering.

- The 1990s have seen an increased emphasis on the 'management' aspects of projects standard of quality and processes just like ISO 9001

4) Explain the SDLC with its phases and diagrams

Software Development Life Cycle (SDLC) is a framework that defines the steps involved in the development of software at each phase. It covers the detailed plan for building, deploying, and maintaining the software.

SDLC defines the complete cycle of development i.e., all the tasks involved in planning, creating, testing, and deploying a Software Product. Every phase of the SDLC life Cycle has its own process and deliverables that feed into the next phase.

Software Development Life Cycle Process:

SDLC is a process that defines the various stages involved in the development of software for delivering a high-quality product. SDLC stages cover the complete life cycle of a software i.e., from inception to retirement of the product.

Adhering to the SDLC process leads to the development of the software in a systematic and disciplined manner.

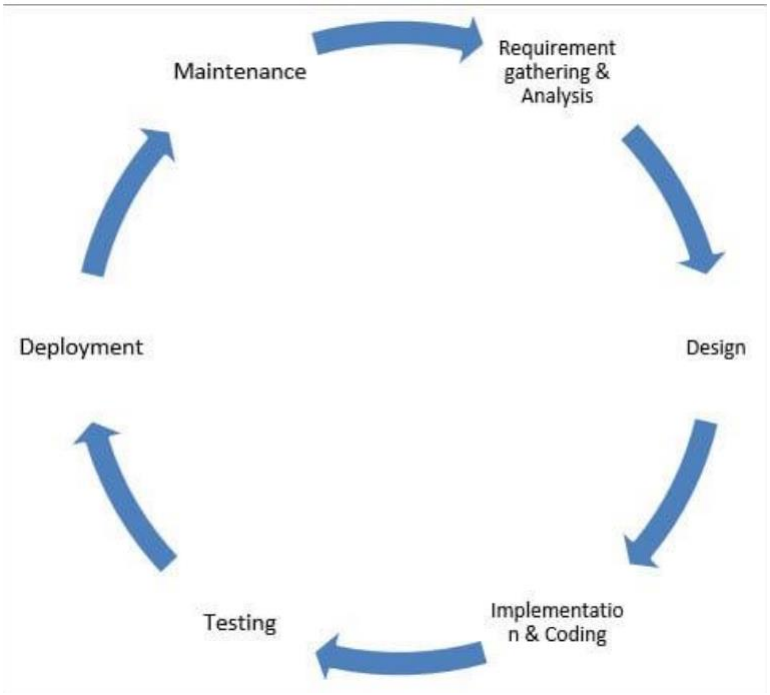
Purpose:

Purpose of SDLC is to deliver a high-quality product which is as per the customer's requirement. SDLC has defined its phases as, Requirement gathering, Designing, Coding, Testing, and Maintenance. It is important to adhere to the phases to provide the Product in a systematic manner. For Example, A software must be developed and a team is divided to work on a feature of the product and is allowed to work as they want. One of the developers decides to design first whereas the other decides to code first and the other on the documentation part.

This will lead to project failure because of which it is necessary to have a good knowledge and understanding among the team members to deliver an expected product.

SDLC Cycle

SDLC Cycle represents the process of developing software. Below is the diagrammatic representation of the SDLC cycle:



SDLC Phases

The entire SDLC process divided into the following SDLC steps:

Phase 1: Requirement collection and analysis

Phase 2: Feasibility study

Phase 3: Design

Phase 4: Coding

Phase 5: Testing

Phase 6: Installation/Deployment

Phase 7: Maintenance

Phase 1: Requirement collection and analysis

The requirement is the first stage in the SDLC process. It is conducted by the senior team members with inputs from all the stakeholders and domain experts in the industry. Planning for the quality assurance requirements and recognition of the risks involved is also done at this stage. This stage gives a clearer picture of the scope of the entire project and the anticipated issues, opportunities, and directives which triggered the project. Requirements Gathering stage need teams to get detailed and precise requirements. This helps companies to finalize the necessary timeline to finish the work of that system.

Phase 2: Feasibility study

Once the requirement analysis phase is completed the next SDLC step is to define and document software needs. This process conducted with the help of 'Software Requirement Specification' document also known as 'SRS' document. It includes everything which should be designed and developed during the project life cycle. There are mainly five types of feasibilities checks:

Economic: Can we complete the project within the budget or not?

Legal: Can we handle this project as cyber law and other regulatory framework/compliances.

Operation feasibility: Can we create operations which is expected by the client?

Technical: Need to check whether the current computer system can support the software

Schedule: Decide that the project can be completed within the given schedule or not.

Phase 3: Design

In this third phase, the system and software design documents are prepared as per the requirement specification document. This helps define overall system architecture. This design phase serves as input for the next phase of the model. There are two kinds of design documents developed in this phase:

High-Level Design (HLD): Brief description and name of each module

An outline about the functionality of every module

Interface relationship and dependencies between modules

Database tables identified along with their key elements

Complete architecture diagrams along with technology details

Low-Level Design (LLD): Functional logic of the modules

Database tables, which include type and size

Complete detail of the interface

Addresses all types of dependency issues

Listing of error messages

Complete input and outputs for every module

Phase 4: Coding

Once the system design phase is over, the next phase is coding. In this phase, developers start build the entire system by writing code using the chosen programming language. In the coding phase, tasks are divided into units or modules and assigned to the various developers. It is the longest phase of the Software Development Life Cycle process. In this phase, Developer needs to follow certain predefined coding guidelines. They also need to use programming tools like compiler, interpreters, debugger to generate and implement the code.

Phase 5: Testing

Once the software is complete, and it is deployed in the testing environment. The testing team starts testing the functionality of the entire system. This is done to verify that the entire application works according to the customer requirement.

During this phase, QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and send back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the business needs of that system.

Phase 6: Installation/Deployment

Once the software testing phase is over and no bugs or errors left in the system then the final deployment process starts. Based on the feedback given by the project manager, the final software is released and checked for deployment issues if any.

Phase 7: Maintenance

Once the system is deployed, and customers start using the developed system, following 3 activities occur

Bug fixing – bugs are reported because of some scenarios which are not tested at all

Upgrade – Upgrading the application to the newer versions of the Software

Enhancement – Adding some new features into the existing software the focus of this SDLC phase is to ensure that needs continue to be met and that the system continues to perform as per the specification mentioned in the first phase

Chp 2

1) What is requirement? What are the types of requirements? Explain with suitable example.

Requirement is a condition or capability possessed by the software or system component in order to solve a real-world problem. The problems can be to automate a part of a system, to correct shortcomings of an existing system, to control a device, and so on. IEEE defines requirement as (1) A condition or capability needed by a user to solve a problem or achieve an objective. (2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents. (3) A documented representation of a condition or capability as in (1) or (2).'

1) Requirements describe how a system should act, appear or perform. For this, when users request for software, they provide an approximation of what the new system should be capable of doing. Requirements differ from one user to another and from one business process to another
   In some cases, a requirement is simply a high level, abstract statement of service that the system should provide or a constraint on the system. At the other extreme it is a detailed formal definition of a system function. Some of the problems that arise during the requirement engineering process

are a result of failing to make a clear separation between these different levels of description. Requirement Engineering is the process of defining, documenting and maintaining the requirements. It is a process of gathering and defining service provided by the system users. The requirements themselves are the descriptions of the system services and constraints that are generated during the requirement engineering process. The requirement engineering is distinguished between the terms :

In some cases, a requirement is simply a high level, abstract statement of service that the system should provide or a constraint on the system. At the other extreme it is a detailed formal definition of a system function. Some of the problems that arise during the requirement engineering process are a result of failing to make a clear separation between these different levels of description. Requirement Engineering is the process of defining, documenting and maintaining the requirements. It is a process of gathering and defining service provided by the system users. The requirements themselves are the descriptions of the system services and constraints that are generated during the requirement engineering process. The requirement engineering is distinguished between the terms

System requirements – It sets out the system functions, services, operational constraints in detail. System requirement means the detail description of what the system should do. The system requirement document should be precise. It should define exactly of is to be implemented. It may be a part of contract between system buyer and software developer.

Example: Library System

User Requirement - Library system shall keep track of all data required by copyright licensing agencies

System Requirement –

- On making a request for a document from the library system the requester shall be presented with a form that records details of user and system mode.
- Library system request forms shall be stored in the system for 5 years from the date of request.
- All library system request forms must be indexed by user, by the name of the material requested and by the supplier of the request.
- Library system shall maintain a log of all requests that have been made to the system

2) What is the classification of the software requirements? Explain with example.

Classification of software requirements

Software requirements is a field within software engineering that deals with establishing the needs of stakeholders that are to be solved by software. **Software system requirements are often classified as functional requirements, non – functional requirements or domain requirements.**

1) **Functional requirements**: In software engineering, a functional requirement defines a system or its component. It describes the functions a software must perform. A function is nothing but inputs, its behavior, and outputs. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform. In some cases, the functional requirements may also explicitly state what the system should not do. These requirements depend on the type of software being developed, the expected users of the software and the general approach taken by the organization when writing requirements.

   Functional requirements in software engineering help you to capture the intended behavior of the system. This behavior may be expressed as functions, services or tasks or which system is required to perform. **Functional requirements indicate what a system must do.**

2) **Non – functional requirements** - A non-functional requirement defines the quality attribute of a software system. They represent a set of standards used to judge the specific operation of a system. Example, how fast does the website load? A non-functional requirement is essential to ensure the usability and effectiveness of the entire software system. Failing to meet non-functional requirements can result in systems that fail to satisfy user needs. On-functional Requirements allows you to impose constraints or restrictions on the design of the system across the various agile backlogs. Example, the site should load in 3 seconds when the number of simultaneous users is > 10000. Description of non-functional requirements is just as critical as a functional requirement. **Non-functional requirements support them and determine**

how the system must perform.

Types of non – functional requirements

These are constraints on the services or functions offered by the system. They include timing constraints, constraints on the development process and standards. Non – functional requirements often apply to the system.

The types of non – functional requirements are:

1) **Product requirements**: Requirements which specify that the delivered product must behave in a particular way e.g., execution speed, reliability, etc.

2) **Organizational requirements**: Requirements which are a consequence of organizational policies and procedures e.g., process standards used, implementation requirements, etc.

External requirements: Requirements which arise from factors which are external to the system and its development process e.g., interoperability requirements, legislative requirements, etc.

Metrics of specifying Non – functional requirements	
Speed	Processed transactions/Event response time, Screen refresh time.
Size	K bytes, Number of RAM chips
Ease of use	Training time, Number of help frames
Reliability	Mean time to failure, Portability of unavailability, Rate of failure occurrence, viability

Robustness	Time to restart after failure, Percentage of events causing failure, Portability of data corruption on failure.
Portability	Percentage of target-dependent statements, Number of target systems

.

3) Write the structure of SRS specified by IEEE.

The software requirements document (sometimes called the software requirement specification or SRS) is the official statement of what the system developers should implement. It should include both the user requirements for a system and a detailed specification of the system requirements. In some cases, the user and system requirements are integrated into a single description. In other cases, the user requirements are defined in an introduction to the system requirements specification. If there are a large number of requirements the detailed system requirements may be presented in a separate document. The requirements documents have a diverse set of users ranging from the senior management of the organization that is paying for the system to the engineers responsible for developing the software. The diversity of possible users means that the requirements has to be a compromise between communicating the requirements to customers, defining the requirements in precise detail for developers and testers including information about possible system evolution

Users of a requirement document	System customers	Specify the requirements and read them to check that they meet their needs. Customers specify changes to the requirements.
	Managers	Use the requirements document to plan a bid for the system and to plan the system development process
	System engineers	Use the requirements to understand what system is to be developed
	System Test Engineers	Use the requirements to develop validation tests for the system

	System Maintenance Engineers	Use the requirements to understand the system and the relationship between its parts

The most widely known standard is IEEE/ANSI 830-1998 (IEEE,1998). This IEEE standard suggests the following structure for requirements documents:

1. Introduction

    1.1 Purpose of the requirements document

    1.2 Scope of the product

    1.3 Definitions, acronyms and abbreviations

    1.4 References

    1.5 Overview of the remainder of the document

2. General description

    2.1 Product perspective

        2.1.1 System interfaces

        2.1.2 User interfaces

        2.1.3 Hardware interfaces

        2.1.4 Software interfaces

        2.1.5 Operations

    2.2 Product functions

    2.3 User characteristics

    2.4 General constraints, Assumptions and dependencies

3. Specific requirements

    3.1 External interface requirements

    3.2 Functional requirements

    3.3 Performance requirements

    3.4 Design constraints

    3.5 Logical database requirement

    3.6 Software system attributes

4. Other requirements

5.Appendices

6.Index

Introduction: This provides an overview of the entire information described in SRS. This involves purpose and the scope of SRS, which states the functions to be performed by the system. In addition, it describes definitions, abbreviations, and the acronyms used. The references used in SRS provide a list of documents that is referenced in the document.

Overall description: It determines the factors which affect the requirements of the system. It provides a brief description of the requirements to be defined in the next section called 'specific requirement'. It comprises the following sub-sections.

Product perspective: It determines whether the product is an independent product or an integral part of the larger product. It determines the interface with hardware, software, system, and communication. It also defines memory constraints and operations utilized by the user

Product functions: It provides a summary of the functions to be performed by the software. The functions are organized in a list so that they are easily understandable by the user:

User characteristics: It determines general characteristics of the users

User characteristics: It determines general characteristics of the users

Assumption and dependency: It provide a list of assumptions and factors that affect the requirements as stated in this document

Specific requirements: These determine all requirements in detail so that the designers can design the system in accordance with them. The requirements include description of every input and output of the system and functions performed in response to the input provided. It comprises the following subsections

External interface: It determines the interface of the software with other systems, which can include interface with operating system and so on. External interface also specifies the interaction of the software with users, hardware, or other software. The characteristics of each user interface of the software product are specified in SRS. For the hardware interface, SRS specifies the logical characteristics of each interface among the software and hardware components. If the software is to be executed on the existing hardware, then characteristics such as memory restrictions are also specified.

Functions: It determines the functional capabilities of the system. For each functional requirement, the accepting and processing of inputs in order to generate outputs are specified. This includes validity checks on inputs, exact sequence of operations, relationship of inputs to output, and so on.

Performance requirements: It determines the performance constraints of the software system.

Performance requirement is of two types: static requirements and dynamic requirements.

Static requirements (also known as capacity requirements) do not impose constraints on the execution characteristics of the system. These include requirements like number of terminals and users to be supported.

Dynamic requirements determine the constraints on the execution of the behavior of the system, which includes response time (the time between the start and ending of an operation under specified conditions) and throughput (total amount of work done in a given time).

Logical database of requirements: It determines logical requirements to be stored in the database. This includes type of information used, frequency of usage, data entities and relationships among them, and so on.

Design constraint: It determines all design constraints that are imposed by standards, hardware limitations, and so on. Standard compliance determines requirements for the system, which are in compliance with the specified standards. These standards can include accounting procedures and report format. Hardware limitations implies when the software can operate on existing hardware or some pre-determined hardware. This can impose restrictions while developing the software design. Hardware limitations include hardware configuration of the machine and operating system to be used.

Software system attributes: It provide attributes such as reliability, availability, maintainability and portability. It is essential to describe all these attributes to verify that they are achieved in the final system.

Other Requirements: Define any other requirements not covered elsewhere in the SRS. This might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and so on. Add any new sections that are pertinent to the project

Chp 3

1) What are the fundamental activities of software process?

Software processes

Software Engineering is defined as the systematic approach to the development, operation, maintenance, and retirement of software. The systematic approach must help to achieve a high quality and productivity (Q&P) of software. A software process (also known as software methodology) is a set of related activities that leads to the production of the software. These activities may involve the development of the software from the scratch, or, modifying an existing system.

A software process specifies the abstract set of activities that should be performed to go from user needs to final product. The actual act of executing the activities for some user needs is a software project and all the outputs that are produced while the activities are being executed are the products.

Software process activities

Real software processes are inter-leaved sequences of technical, collaborative, and managerial activities with the overall goal of specifying, designing, implementing, and testing a software system.

The four basic process activities of specification, development, validation, and evolution are organized differently in different development processes. In the waterfall model, they are organized in sequence, whereas in incremental development they are interleaved.

1) Software specification

The process of establishing what services are required and the constraints on the system's operation and development.

Requirements engineering process:

Feasibility study: is it technically and financially feasible to build the system?

Requirements elicitation and analysis: what do the system stakeholders require or expect from the system?

Requirements specification: defining the requirements in detail

Requirements validation: checking the validity of the requirements

2) Software design and implementation

The process of converting the system specification into an executable system.

Software design: design a software structure that realizes the specification;

Implementation: translate this structure into an executable program;

The activities of design and implementation are closely related and may be interleaved.

Design activities include:

Architectural design: identify the overall structure of the system, the principal components (sometimes called sub-systems or modules), their relationships and how they are distributed.

Interface design: define the interfaces between system components.

Component design: take each system component and design how it will operate.

Database design: design the system data structures and how these are to be represented in a database.

3) Software validation

Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer.

Validation: are we building the right product (what the customer wants)?

Verification: are we building the product, right?

V & V involves checking and review processes and system testing. System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.

Testing is the most used V & V activity and includes the following stages: **Development or component testing**: individual components are tested independently; components may be functions or objects or coherent groupings of these entities.

System testing: testing of the system, testing of emergent properties is particularly important

Acceptance testing: testing with customer data to check that the system meets the customer's needs.

4) Software evolution

Software is inherently flexible and can change. As requirements change through changing business circumstances, the software that supports the business must also evolve and change. Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new**.**

2) Explain the term software project with an example

3) Explain the difference between software process and software project

A process is a sequence of steps performed for a given purpose. As mentioned earlier, while developing (industrial strength) software, the purpose is to develop software to satisfy the needs of some users or clients, as shown in Figure 2.1. A software project is one instance of this problem, and the development process is what is used to achieve this purpose.

So, for a project its development process plays a key role—it is by following the process the desired end goal of delivering the software is achieved. However, as discussed earlier, it is not sufficient to just reach the final goal of having the desired software, but we want that the project be done at low cost and in low cycle time, and deliver high-quality software. The role of process increases due to these additional goals, and though many processes can achieve the basic goal of developing software in Figure 2.1, to achieve high Q&P we need some "optimum" process. It is this goal that makes designing a process a challenge. We must distinguish process specification or description from the process itself. A process is a dynamic entity which captures the actions performed. Process specification, on the other hand, is a description of process which presumably can be followed in some project to achieve the goal for which the process is designed. In a project, a process specification may be used as the process the project plans to follow. The actual process is what is actually done in the project. Note that the actual process can be different from the planned process, and ensuring that the specified process is being followed is a nontrivial problem. We will assume that the planned and actual processes are the same and will not distinguish between the two and will use the term process to refer to both. A process model specifies a general process, which is "optimum" for a class of projects. A process model is essentially a compilation of best practices into a "recipe" for success in the project. In other words, a process is a means to reach the goals of high quality, low cost, and low cycle time, and a process model provides a process structure that is well suited for a class of projects. A process is often specified at a high level as a sequence of stages. The sequence of steps for a stage is the process for that stage, and is often referred to as a subprocess of the process.

4) Write a short note on component software

process

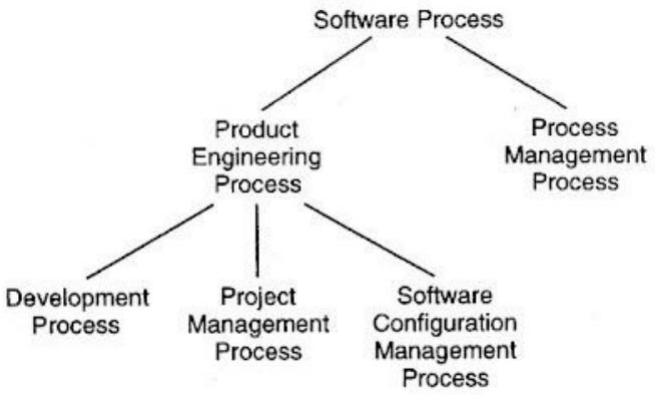
A process is the sequence of steps executed to achieve a goal. Since many different goals may have to be satisfied while developing software, multiple processes are needed. Many of these do not concern software engineering, though they do impact software development. These could be considered non-software process. Business processes, social processes, and training processes are all examples of processes that come under this. These processes also affect the software development activity but are beyond the purview of software engineering.

There are clearly two major components in a software process—a development process and a project management process. The development process specifies all the engineering activities that need to be performed, whereas the management process specifies how to plan and control these activities so that cost, schedule, quality, and other objectives are met. Effective development and project management processes are the key to achieving the objectives of delivering the desired software satisfying the user needs, while ensuring high productivity and quality.

During the project many products are produced which are typically composed of many items (for example, the final source code may be composed of many source files). These items keep evolving as the project proceeds, creating many versions on the way. As development processes generally do not focus on evolution and changes, to handle them another process called software configuration control process is often used. The objective of this component process is to primarily deal with managing change, so that the integrity of the products is not violated despite changes

These three constituent processes focus on the projects and the products and can be considered as comprising the product engineering processes, as their main objective is to produce the desired product. If the software process can be viewed as a static entity, then these three component processes will suffice. However, a software process itself is a dynamic entity, as it must change to adapt to our increased understanding about software development and availability of newer technologies and tools. Due to this, a process to manage the software process is needed.

The relationship between these major component processes is shown in Figure 2.2. These component processes are distinct not only in the type of activities performed in them, but typically also in the people who perform the activities specified by the process. In a typical project, development activities are performed by programmers, designers, testers, etc.; the project management process activities are performed by the project management; configuration control process activities are performed by a group generally called the configuration controller; and the process management process activities are performed by the software engineering process group (SEPG).


Software Process
Product Engineering Process
Process Management Process
Development Process
Project Management Process
Software Configuration Management Process