CHAPTER 1

UNIT I

Introduction to Software Engineering

Syllabus

Introduction : What is software engineering ? Software Development Life Cycle, Requirements Analysis, Software Design, Coding, Testing, Maintenance etc.

1.1 Introduction

- Computers are becoming an unavoidable necessity or say the part and parcel of our life.
- We do most of our day's work using computers and digital devices – to book railway tickets or movie tickets, search for some books on Amazon, bank transactions, and any task you name it is done using computers.
- Therefore it is very important to build these computerized systems in an effective way.
- Building such systems requires certain technical as well as communication skills and capabilities to understand and follow a planned and systematic procedure.

1.2 Software

Q. What is software?

- Software is a set of instructions to acquire inputs and to process them to produce the desired output in terms of functions and performance as determined by the user of the software. It is developed to handle an Input-Process-Output system to achieve predetermined goals.

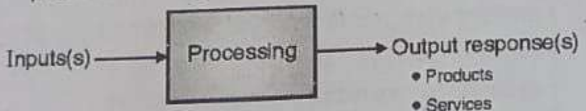

Fig. 1.2.1 : Basic diagram of a software

Q. What does Software encompass ?

Software encompasses of :

1. Instructions (Computer programs)
2. Documents (Describe programs)
3. Architecture including Data Structures (Enable programs)

- Examples : Websites, programs or video games etc.

Software - a product and a vehicle :

"Software plays a dual role - both as a product and a vehicle that delivers a product".

A. Software is a product

1. Delivers computing potential.
2. Produces, manages, acquires, modifies, displays, or transmits information.(e.g., software is in a cellular phone or in any computer).

B. Software is a vehicle for delivering a product

1. Control other programs (e.g. An operating system)
2. Effect communications (e.g. Networking software)
3. Help build other software (e.g. Software tools)

1.2.1 Characteristics of Software

Q. What are the characteristics of software ?

- Software means anything which is not hardware but which is used with hardware, such as windows OS (is system software) in computer (is hardware).
- You will get more precise idea about the characteristics of software and how it differs from hardware from the Table 1.2.1.

Table 1.2.1 : Characteristics of software and how it differs from hardware

Sr. No.	Software	Hardware
1.	It is developed or engineered.	It is manufactured.
2.	It doesn't wear out as it is not prone to environmental problems.	It wears out as the time passes due to the affects of dust, vibration, temperature extremes and many such environmental problems.

Sr. No.	Software	Hardware
3.	There are no software spare parts which can be used to replace the software.	When hardware fails, it can be replaced by spare parts.
4.	Software is untouchable. It is the code or instructions that tell a computer/hardware how to operate. It has no substance.	Hardware is a physical device something that you're able to touch and see.
5.	Software is usually generic but it can also be custom built (developed according to the customer specification).	It is manufactured or assembled by using the existing components.
6.	Software is invaluable as it can be installed in any hardware.	Hardware has no value without software in it. If computers (h/w) had no operating system (s/w), then no customer will buy it.
7.	Example 1 : Linux, Windows any operating system that allows you to control your computer. Example 2 : Internet browsers, video games, applications like Payroll etc.	Examples : Disks, disk drives, display screens, keyboards, printers and chips.

1.2.2 Classes of Software

Q. Describe the various categories of software.

Software is classified into two types of classes :

1. Generic
2. Customized

Table 1.2.2 : Difference between generic software and customized software

Sr. No.	Generic software	Customized software
1.	Designed for broad customer market.	Designed for specific business purposes.
2.	Is open to market and its specifications are designed by the programmer.	Is an s/w and its specifications are designed according to a particular firm or organization; it is not open for all.
3.	Examples : ERP/CRM, CAD/CAM packages, OS, system software.	Examples : Legacy systems, Process control systems, traffic management system and hospital management system.
4.	Generic software development is done for general purpose audience.	Custom software development is done to satisfy a particular need of a particular client.
5.	Requirements and specifications of this software are managed by the developer.	Managed by the customer and influenced by the practices of that industry.
6.	General purpose software development is tough as compared with custom made by the design and marketing point of view.	By Buyer's point of view, he prefers to have a custom made application developed rather than buying a general purpose software as he gets the application done that exactly matches his requirements.
7.	In general purpose application design and development, you need to imagine what an end-user requires. Market Surveys and general customer demand analysis may help in such designs.	In custom made application, you have a specific end – user. Understanding his need and analyzing it to get the best out of it helps in such designs.

1.2.3 Changing Nature of Software

Software is classified into 7 categories as below :

1. **System software**

It directly interacts with computer hardware. It is generic software.

web design. It allows communication, information sharing, interoperability, user-centered design and collaboration on www.

- Third-generation Web application is a powerful business tool for organizations in their Electronic Commerce (EC) efforts.

7. **Artificial intelligence software**

- This software makes use of fuzzy logic to solve complex problems and require specific analysis and interpretation of the problem to solve it.
- **Examples :** Robotics, expert systems, artificial neural networks and computer games. All these software's can run either in real-time mode or offline mode. This software's can be shared free or charged by usage.
- Thus, they are also called shareware and freeware. Some of these types of software are run on desktop computers, others on mainframe and mini computers, some are exclusively designed for web, network or the internet.

Q.2

1.3 What is Software Engineering ?

Q. Explain software engineering concept in detail.

Definition

- Software Engineering is a systematic, scientific and disciplined approach towards the development, functioning and maintenance of the software.
- As described in the above definition, the systematic and scientific approach in software engineering can be defined as :
  - o Understanding of customer requirements both by customer and developer.
  - o Use of structured methodology to gather customer requirements and its analysis to arrive at the Software Requirements Specification(SRS)
  - o Right estimation of resource, efforts and costs.
  - o Use of developmental and testing methodology to ensure the quality of software.
  - o Ease of installation, demonstration and implementation of software by the customer and users.
- Such an Engineering approach is required to ensure that the software is designed with the correct choice of technology and architecture to :
  - o Achieve customer satisfaction
  - o Ensure on-time delivery
  - o Be developed within the budget cost
  - o Provide ease of maintenance to meet changing requirements.
- Else it leads to :

1. Unreliable and poor quality software development.
2. Late delivery to the customer.
3. Difficulty in maintenance
4. Lot of expenses in development process ultimately expensive software.

- **Primary goal** of software engineering: To provide quality software at low cost on planned delivery date. Software engineering involves various phases such as project planning, systematic analysis and design, testing and maintenance.

Basic Objectives of Software Engineering

Q. What are the basic objectives of software engineering ?

1. Define software development plan
2. Manage software development activities
3. Design the proposed product
4. Code/develop the product
5. Test the product modules
6. Integrate the modules and test the system as a whole
7. Maintain the product

- Software Engineering is the part of a larger subject called *Systems Engineering* which considers issues like hardware platform, operating system, and interoperability between platforms, performance, scalability and upgrades to develop the software.
- While considering these issues, it uses Computer Aided Software Engineering (CASE) tools (e.g. ArgoUML, Case Studio 2, Magic Draw), software quality testing tools, code generators (e.g. XMLSpy, UModel), and report writers. Thus, a good software engineer must know how to use these tools to develop quality software.

Key challenges of Software Engineering

- **Heterogeneity** : To use the development techniques that can cope with diverse platforms and execution environments.
- **Delivery** : To use development techniques that lead to faster software delivery.
- **Trust** : To use development techniques that can build trust in users' opinion.

1.3.1 History of Software Engineering

Software engineering era can be divided into three periods :

1. The Pioneering Era (1955-1965)

- New digital computers are used every year making the existing software outdated due to which programmers were required to rewrite all their software programs which would be suitable on these new computers. So as to develop programs suitable for new machines, the high-order languages like FORTRAN, COBOL, and ALGOL were developed.
- Programmers had no desktop/personal computers, therefore always had to go to the "machine room" for programming.
- Programming was done by putting punched cards for input into the machine's card reader and waiting for results to come back on the printer.
- In the era of such a slow process of programming, it was almost impossible to make predictions of a project's completion date.
- Hardware vendors sold systems software for free with hardware because hardware has no meaning without software.

2. The Stabilizing Era (1965-1980)

- Then came IBM 360. It put an end to the era of faster and cheaper computers emerging every year or two. Software people could then spend time writing new software instead of updating the old. The 360 also combined scientific and business applications onto one machine. It offered both binary and decimal arithmetic.
- The programmer had to write the program in a whole new language i.e. Job Control Language (JCL) to tell the computer and OS what to do.
- The demand for programmers exceeded.
- As the software field stabilized, software became a corporate asset and its value became huge.
- "Structured Programming" was developed in the middle of this era.

3. The Micro Era (1980-Present)

- The price of computers had dropped by now by which most of the programmers could have a computer on his desk.
- The commonly-used programming languages today are 15 years old.

Q. 4

1.4 Software Engineering - A Layered Technology

Q. Explain the layered technology of software engineering.

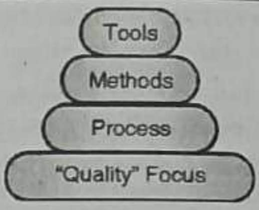

Fig. 1.4.1 : Layers of software engineering

1.4.1 Quality Focus

- Software Engineering rests on an organizational commitment to *quality*, which leads to continuous improvements in the Software engineering process.
- Focus on quality is always the primary goal of software engineering. In short, we can say that the software is qualitative :
  - If it is read and understood easily,
  - If it is modifiable and accommodates new requirements,
  - If it has customer satisfaction and is completed in time within budget.
- **The primary goal of any software process** : High quality
- **Remember :** High quality = project timeliness
  Why ? Less rework!

1.4.2 Software Process

- When building a product, it's important to go through some predictable steps i.e. called *process* that helps you to create a timely and high quality product.
- A *process* is the foundation of software engineering. It defines a framework that must be established to ensure effective delivery of Software Engineering technology. The key use of a *process* is :

- Producing models, documents, data, forms and reports.
- Establishing milestones.
- Software Quality Assurance (SQA).
- Software Configuration (change) Management (SCM).

1.4.3 Software Engineering Methods

- Method is an ordered way of developing a software. This includes the suggestions for the process to be followed, the notations to be used, the rules governing the system descriptions and the design guidelines.
- It provides the technical "how to' s" for building a software. It includes how to implement the following generic processes :
  1. Communication
  2. Requirements analysis
  3. Design
  4. Program construction
  5. Testing
  6. Maintenance

1.4.4 CASE Tools

- They provide automated or semi-automated support for the process and the methods. For example; CASE tool.
- CASE represents Computer Aided Software Engineering tool used in software development process.
- A CASE tool is a computer-aided product specially designed to support the software engineering activities within a software development process.

Examples of Case tools

- Code generation CASE tools - Visual Studio .NET
- Code analysis CASE tools -Borland Audits
- CASE tools used for development of data models -UML editors
- Cleaning up code CASE tools -Refactoring tools
- Bug tracker CASE tool
- Version control CASE tool -CVS, etc.

Benefits

1. Improves the productivity
2. Generates small parts of code automatically
3. Improves software quality
4. Can be integrated with other tools say, with code editor to work with coding.

CO5

1.5 Core Principles of SE Practice

Q. State the core principles of software engineering practice.

- Principle means an important law or assumption required in a system of thought.
- Principles help us establish some standards or rules for solid software engineering practice.

David Hooker has proposed 7 core principles that focus on software engineering practice as listed below :

1. **Provide Value to Your Users (The Reason It All Exists) :** Before preparing specifications, before determining the hardware platforms or development process, be sure that this adds real value to the system. If it doesn't then don't do it because the software system exists only for one reason i.e. to provide value to its users.
2. **Keep It Simple, Stupid (KISS) :** The software designing should be as simple as possible. But, simple doesn't mean discarding any internal features and neither has it meant 'quick and dirty'. Simple means - the software is less error-prone, easy to understand and maintain.
3. **Maintain the Vision :** A clear vision and conceptual integrity is essential to the success of a software project.
4. **What You Produce, Others will Consume :** The software engineer should always specify, design, and implement the software product knowing that someone else will have to understand what you are doing. The users of his product are potentially large. He should design and code, keeping the end users in mind because making the end user's job easier adds value to the system.

5. **Be Open to the Future :** A system with long life time has more value. Thus, a system should be developed so that the entire system can be reused and this is possible only if it is adaptable to the changes.
6. **Plan ahead for Reuse :** Reuse save time and effort. Use of object oriented technologies lead to the benefit of reuse of code and designs. Reuse reduces the cost and increases the value of both the reusable components and the systems into which they are incorporated.
7. **Think before You Do :** Always make clear and complete planning before doing any action. This will produce better results. When you think about something, you are more likely to do it right or you may at least gain knowledge to how to do it right the next time.

1.6 Software Development Life Cycle (SDLC)

Q. State the SDLC phases in brief.

- The Systems Development Life Cycle (SDLC) is a conceptual model used in software project development that describes the phases involved in developing an information system.
- It includes all the details right from the initial feasibility study through the maintenance of the completed application.
- Various SDLC methodologies have been designed to help the developers including the **waterfall model which was the original SDLC method**, RAD, JAD and spiral model.
- Uses of methodologies differ as the types of projects differ. But in the final analysis, the most important factor that make the project successful is how closely it confirms to the user requirements.
- In general, an SDLC methodology follows these steps:

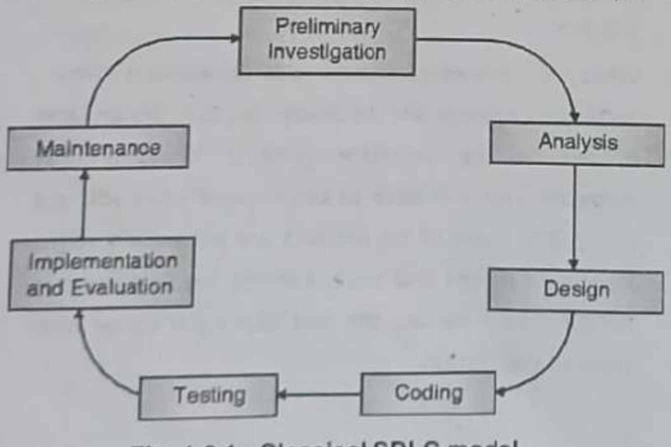

Fig. 1.6.1 : Classical SDLC model

1. **Preliminary Investigation :** This activity includes the activities of requirement gathering, analyzing and designing the requirements in an effective manner understandable by the client. This phase is accomplished by conducting formal and informal interviews with the clients, end users.
2. **Requirement Analysis :** Once the requirements of the proposed project are defined, the requirement Analysis is done to enable the software engineers to better understand the problem they will be working on. The analysis is performed using set of analysis techniques such as decision trees, decision tables, and etc.) that lead to understand what impact the software will have on the business, what the customer wants and how end users will interact with the software. Thus, the involvement of both- software engineers and end users is needed in requirement analysis.
3. **Design :** The proposed system is designed using the design models such as functional decomposition diagrams, Data Flow Diagrams (DFD), Entity Relationship Diagrams (ERD) or any Unified Modeling Language (UML) diagrams.
4. **Coding :** The new components and programs are developed. This phase is just implementing the design in to programming language that means it actually develops the proposed system.
5. **Testing :** This phase tests whether the software gives the valid and expected output or not, are there any errors or bugs in the development of software or if any changes are required. Testing is required because an erroneous and improper development of software causes financial loss, time loss and loss of efforts put in development process. If the errors are not detected and eliminated, it increases the estimated cost of the system development.

   The cost of fixing a bug (error) and making the required changes in early phases of software development is less as compared to the same detected in later phases. This is because making changes in earlier phases is easy and if the bugs are detected in later phases, it becomes difficult to repeat the previous phases once again in order to make the required changes. Thus the cost goes high.

CHAPTER 2

UNIT I

Software Requirements

Syllabus

Software Requirements : Functional and Non-functional requirements, User Requirements, System Requirements, Interface Specification, Documentation of the software requirements.

2.1 Software Requirements

- A requirement is about 'what' rather than 'how'.
- A requirement is a condition needed by a user to solve a problem or achieve an objective.
- **Example :** A requirement for a car could be that the maximum speed to be at least 120 mph, fuel tank should have at least 5 litres volume and so on.
- Defining *requirements* is the first step in the development of a system. But, in many situations, enough care is not taken in establishing correct requirements.
- This causes problems, later in the development life cycle, and more time and money is spent in fixing these problems.
- Thus, it is necessary that requirements are established in a systematic way to ensure their accuracy and completeness.

☆ 2.2 Functional and Non-Functional Requirements

- Functional requirements describe the "processing" of the information and non-functional requirements support the business standards and the business environment.
- Functional requirements are the services that a system should provide i.e. the do's and don'ts of a system when a particular input is given.
- Non-functional requirements are the constraints applied on the services of the system. Non-functional requirements may be the color of the user interface, speed of input/output communication, security of the information.

Q: 6

Difference between functional and non-functional requirements

Q. Differentiate functional and non-functional requirements.

Table 2.2.1 : Difference between functional and non-functional requirements

Sr. No.	Functional Requirements	Non-functional Requirements
1.	Describes Product features	Describes Product properties as the quality attributes.
2.	Describes what the product should do i.e. the actions/work/services of the software.	Describes capabilities of the product i.e. how the software should behave to meet the user needs.
3.	Describe the services that a system should provide	Describes the design and implementation constraints on the services and the external interface which a product must have.
4.	Describe 'what' the system should do in a particular condition. **Example :** A bank software has functions - open acc., close acc, withdraw, loan claim etc.	Describe 'how' the system should work so as to be user friendly and secured and also describes the throughput of the system i.e. the mean time between request and response of a page. **Example :** Performance, reliability, portability, security.

2. **SRS reduces the development effort**. The work of preparing SRS forces various stakeholders i.e. various concerned groups in the customer's organization to think thoroughly of all their requirements before the project design begins, thus reducing the efforts needed for redesigning, recoding, and retesting. Careful inspection of the requirements specified in SRS can reveal the left-outs, misinterpretations and inconsistencies early in the development cycle; hence it becomes easier to correct these problems.
3. **SRS forms a base for cost estimation and project scheduling**. As complete description of the product to be developed is specified in the SRS, it helps in estimating the project costs and can be used to obtain approval from the customer for the decided price estimates.
4. **SRS provides a baseline for verification and validation**. Software development team can develop their verification and validation plans or say, test plans much more effectively from a well-prepared SRS document. As SRS provides a baseline against which requirement confirmation can be measured.
5. **SRS facilitates transfer of new software to new environments**. SRS makes it easier to transfer the software product to new clients or new hardware machines. Therefore, developers feel easier to transfer the software to new clients and customers feel easier to transfer the software on other systems of their organization.
6. **SRS serves as a basis for software improvement.** SRS describes the product features and not the process of project development; therefore, it serves as a basis for enhancements by allowing the developers to do any later modification if required on the finished product. But then, SRS needs to be altered in that case i.e. SRS forms a base for continuous product evaluation.

2.5.3 Characteristics (Quality Criteria) of a Great SRS

Q. Describe the quality criteria of a good SRS.

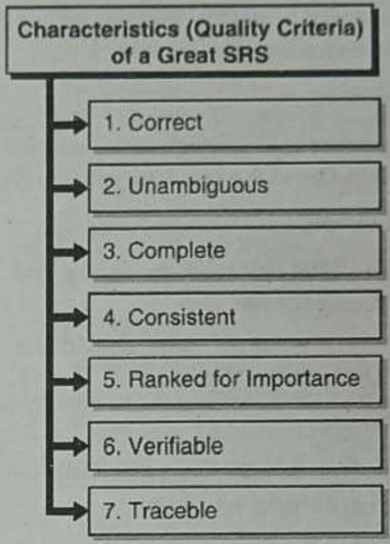

Fig. 2.5.1 : Characteristics (Quality Criteria) of a Great SRS

1. **Correct**

   A set of requirements is correct if and only if every requirement stated therein represents something required of the system to be built." Davis (1993)

2. **Unambiguous**

   The reader of a requirement statement should be able to draw only one interpretation of it.

- **Incorrect Example** – "The system should not accept password longer than 8 characters".
- The above stated requirement can have multiple interpretation as given below :
- The system should not allow user to enter more than 8 characters in the password field.
- The system should truncate the entered data to 8 characters.
- The system should display an error message if user enters more than 8 characters in the password field.
- **Correct Example** – "The system should not accept passwords longer than 8 characters in the password field. If the user enters more than 8 characters while entering the password, an error message should prompt the user to correct the same."

3. **Complete**

   No requirements should be missing. A complete requirement leaves no room for guessing. Include all significant requirements, whether related to functionality, performance, design constraints, attributes, or external interfaces.

4. **Consistent**

   A consistent requirement does not conflict with other requirements in the requirement specification.

Incorrect Example :

- REQ1 – "The birth date should be entered in mm/dd/yyyy format".
- REQ2 - "The birth date should be entered in dd/mm/yyyy format".

Correct Example :

- REQ1 – "For the US users the birth date should be entered in mm/dd/yyyy format"
- REQ2 – "For the French users the birth date should be entered in mm/dd/yyyy format".

5. **Ranked for Importance**

   Requirements should be achievable. But sometimes, few requirements are not attainable.

6. **Verifiable**

   Don't put in requirements like – "It should provide the user a fast response" or "The system should never crash". Instead, provide a quantitative requirement like: "Every key stroke should provide a user response within 100 milliseconds."

7. **Traceable**

   Each requirement should be expressed only once and should not overlap with another requirement. Every requirement has a unique identification number so that requirements can be easily traced throughout the specification.

2.5.4 IEEE Standard SRS Format

Q. Explain the format and features of a good SRS.

1. INTRODUCTION

   PRODUCT OVERVIEW

   (Product description)

   PURPOSE

   (Purpose of the product i.e. the services expected from the product)

   SCOPE

   Write the benefits, objectives, and goals.

   AUDIENCE

   (lists the users of SRS such as developers, project managers, marketing staff, users, testers, and documentation writers.)

   DEFINITIONS, ACRONYMS AND ABBREVIATIONS

   (defines all the terms necessary to properly interpret the SRS)

   CONVENTIONS

   (describes the standard conventions followed while writing this SRS such as fonts or special significant highlights)

   REFERENCES

   (lists the reference documents or web page address used as reference)

2. OVERALL DESCRIPTION

   A. PRODUCT PERSPECTIVE

   (designs that describe the functional and data flow of the product)

   B. PRODUCT FUNCTIONALITY

   (designs such as DFDs that describe major functions of the system)

   C. USER CHARACTERISTICS

   (Identify various users who will be using this product. Differentiate them based on frequency of use, subset of product functions used, technical expertise, security or privilege levels, educational level, or experience.)

   D. OPERATING ENVIRONMENT

   (Designs that describe that shows the major components of the overall system, subsystem interconnections, and external interface.)

   E. DESIGN AND IMPLEMENTATION CONSTRAINTS

   Describe the issues of the developers such as - hardware limitations, interfaces to other applications, specific technologies, tools, and databases to be used, language requirements, communications protocols, security considerations, design conventions.

CHAPTER 3

UNIT I

Software Processes

Syllabus

Software Processes : Process and Project, Component Software Processes.

3.1 Introduction

- When building a product, it's important to go through some predictable steps i.e. called *process* that helps you to create a timely and high quality product.
- A *process* is the foundation of software engineering. It defines a framework that must be established to ensure effective delivery of Software Engineering technology.

The key use of a process

- Producing models, documents, data, forms and reports.
- Establishing milestones.
- Software Quality Assurance (SQA).
- Software Configuration (change) Management (SCM).

3.2 Process and Project Q3

- A software project is an instance of development process.
- Development process leads the project from user needs to software.

3.2.1 Software Process

Q. Define the terms : Software Process.

- **Process** : It is a sequence of steps performed to achieve some goal
- **Software Process** : It is the sequence of steps performed to produce a software with high quality within estimated budget and schedule.

3.2.2 Software Project

Q. Define the terms : Software Project.

- **Project** : It is to build a software system within estimated cost and schedule and with high quality which satisfies the customer.
- Suitable software process is required to reach goals. This process helps in achieving the highest quality software.

3.3 Components of Software Processes

Q. Define the terms: Components of software process.

Components of Software process are depicted in the Fig. 3.3.1.

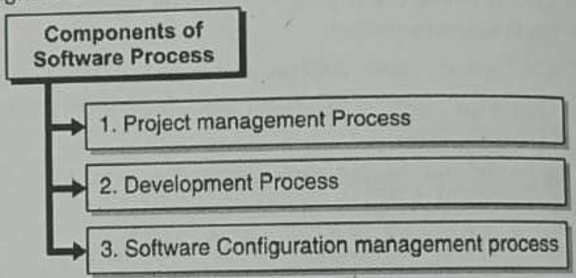
Components of Software Process
1. Project management Process
2. Development Process
3. Software Configuration management process

Fig. 3.3.1 : Components of Software Process

1. **Project management process**
   - It focuses on planning and controlling the development process.
2. **Development process**
   - It focuses on development and quality steps needed to engineer the software. Development process is the heart of software process other processes revolve around it.
   - Development process is executed by different people : developers execute Eng. Process project manager executes the management. process.
3. **Software configuration management process**

   It manages the evolution of artifacts

   - **Change Management Process** : How changes are incorporated
   - **Process Management Process** : Management of processes themselves
   - **Inspection Process** : How inspections are conducted on artifacts

4.2 Waterfall Model (Oldest SDLC) Q.8

Q. Explain all phases of water fall model

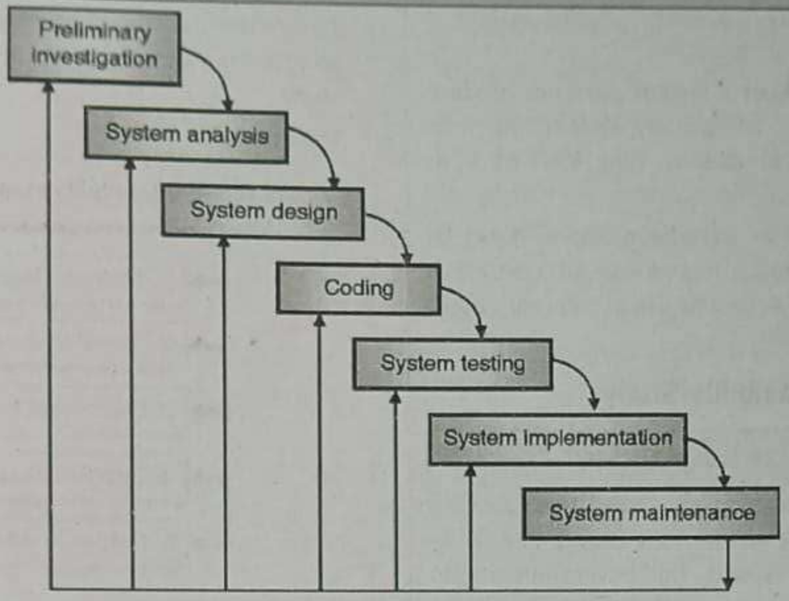

Fig. 4.2.1 : Waterfall model

- This linear waterfall model was first proposed by 'Winston Royce'. It suggests systematic sequential approach for *software development.*
- It is the oldest paradigm for software engineering i.e. it is the oldest ***software development life cycle***.

4.2.1 Preliminary Investigation

- Preliminary investigation means total inspection of the existing system i.e. clear understanding of the system.
- Its basic task is to find out real problem of the system with causes and complexity of the problem. Its secondary but very important task is to find out all possible solutions to solve that problem and according to that which solution is feasible for the system in terms of technology, cost, operational. Its last task is to mention all benefits can be expected after problem is solved.
- So this phase is divided into three main goals as follows :

1. Problem identification
2. Possible and feasible problem solution i.e. Feasibility study.
3. Expected benefits after the problems are solved.

4.2.1(A) Problem Identification (Problem Analysis)

It requires to completely investigate the environment of the system. Generally it requires studying two environments: Internal environment and external environment, which are listed below :

Sr. No.	Internal Environment	External Environment
1	Company Management	Customers
2	Employees of all departments	Management consultant
3	Internal auditors	External auditors
4	Data Processing department	Government Policies
5	Financial Reports	Competitions

There are normally seven types of problems encountered in the system :

1. **Problem of Reliability :** If system may not work properly or same procedures give different (i.e. unreliable) result.
2. **Problem of Validity :** Reports contain misleading information.
3. **Problem of Economy :** System is costly to maintain.

Sr. No.	Classic Life Cycle Model	Prototyping Model
3.	Testing is not conducted from the initial phases of the SDLC – therefore when errors are detected after the coding phase it costs a lot for refining the product.	Building and refining of product is involved before the actual engineering process – therefore, much cost is not involved
4.	Included Manual coding	Uses readymade tools such as CASE tools for coding
5.	Communication with the users is done at the start i.e. while requirement gathering and then while testing phase. No ongoing communication with the users is encouraged	develops the product through ongoing communication with the user

4.4.2 The Spiral Model Q.9

Q. What does the radius imply in the spiral model.

Q. Explain the spiral models.

The Spiral model is an evolutionary and iterative software process model and it is proposed by Boehm.

Features

- Each ***cycle*** around the spiral may be like a phase.
- The ***radius*** of the spiral represents the cost incurred so far in the process.
- Spiral model is represented in the form of ***Cartesian diagram with 4 quadrants*** that represents 4 phases of software development.

Main Objectives of Spiral Model

- It provides controlled and systematic aspects of the linear sequential model.
- It uses rapid development potential of incremental versions of the software.
- It finds all risks in the project.
- It finds future risks also.

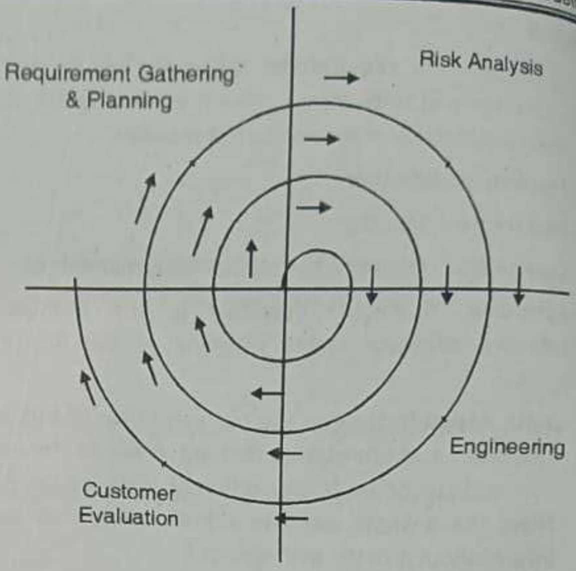

Fig. 4.4.2 : Basic spiral model

Method

- Requirement Gathering includes :
  - Interaction with customer and user.
  - Problem identification regarding existing system.
  - Deciding product specification.
  - Deciding objectives of the system.
  - Planning decides the project plan. It includes
  - Cost measurement of the system.
  - Deciding schedule of the system.
  - Collecting feedback from the user.
  - Adjusted and planned number of iterations required to complete software.
- **Risk Analysis** task is involved in each of the iterations of the project. It includes:
  - Identification of risk areas
  - Identification of technical as well as managerial risks
  - Measuring cost related risk
  - Efforts are taken to resolve the risk
- **Engineering** phase immediately designs the prototype of the proposed system depending upon the objectives decided in the planning phase.

- It also converts design into coding and does the needful testing. It then shifts to *deployment phase* which includes :
  - o Implementing the developed system
  - o Gives training to users
  - o Do the Verification
  - o Collect feedback
- **Customer Evaluation :** The product is then evaluated by the customers or end-users to find if any changes are required. If so, then the above phases are repeated again in loops.

Difference between Spiral Model and Waterfall Model

Table 4.4.2 : Comparison between Waterfall and Spiral Model

Sr. No.	Waterfall Model	Spiral Model
1.	Process flows from top to bottom like a flow of water from a hill to ground. Flow of water can't be reversed - similarly any new changes cannot be incorporated in the middle of the project development.	Best suitable for projects associated with risks.
2.	Process goes to the next phase only after the completion of the previous phase. Here, end user feedback is not taken into consideration for any change in SRS. This will result in restarting the work from beginning.	Each and every step goes through testing which makes it easy to recover any error and fix it then and there itself. In this model we don't have to start work from beginning.
3.	Purely a pre planned /strategic in nature.	Used to build a product which doesn't have adequate requirement gathering.
4.	Focuses more on requirement gathering.	Focuses more on risk analysis which is not much considered in waterfall model.
5.	Customer feedback is not considered at every step of project development.	Customer feedback is considered at every step of project development.

4.4.2(A) Win-Win Spiral Model

- Win-Win Spiral Model is based on Theory W which is a management theory based on "making winners of all of the system's key stakeholders as a necessary and sufficient condition for project success".
- Stakeholder is a person, group, organization, or system who directly or indirectly affects or gets affected by the system.

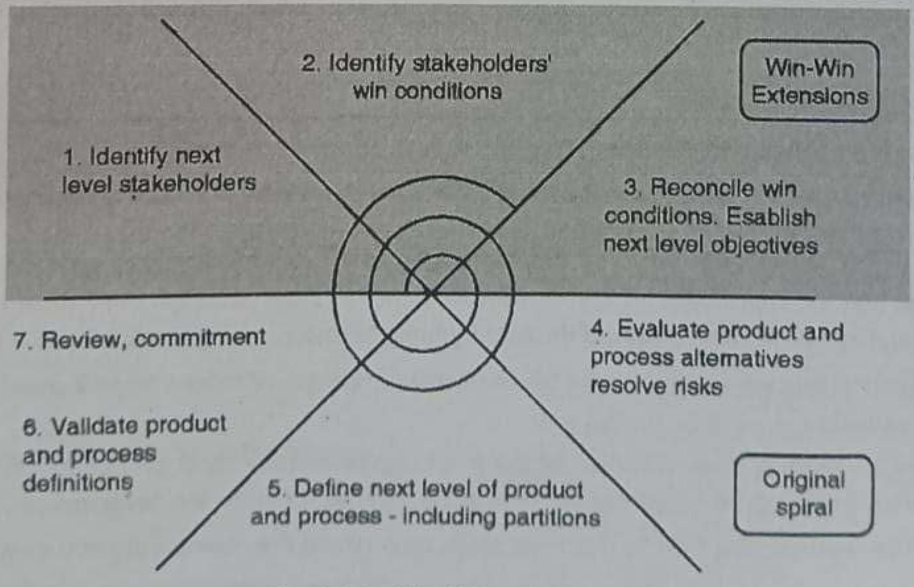
2. Identify stakeholders' win conditions
Win-Win Extensions
1. Identify next level stakeholders
3. Reconcile win conditions. Esablish next level objectives
7. Review, commitment
4. Evaluate product and process alternatives resolve risks
6. Validate product and process definitions
5. Define next level of product and process - including partitions
Original spiral

Fig. 4.4.3 : Win-Win spiral model

Drawbacks of the Incremental Model

1. All the tasks are not decided in first phase so there may be some problem with designing phase. That means, overall design of the system is not so good.
2. It may produce software, which requires large space in memory.
3. Some times, speed of software is also slow.
4. Delivery date of the product could not be decided.
5. New levels of increments may require new hardware because old hardware is involved with old increments.
6. Some times, it produces same code in different module because of partial functionality.
7. Some times, problem is not identified properly.
8. Quality of software or product is poor.

4.3.2 RAD Model AQ10

Q. Write short note on RAD.

- RAD is Rapid Application Development. It is high speed adaptation of waterfall model. It is the example of incremental software process model.
- The main disadvantage of incremental model is - the quality of product is very poor as well as it also requires more time to develop. Therefore, RAD model is only designed to produce a good quality product in very short duration of time.

Method

1. It also adapts generic framework activities like waterfall model that means five phases.
2. This allots sufficient time for study of the existing system.
3. It accepts sufficient number of staff for development of the system.
4. Staff is distributed into various teams (i.e. senior programmer, junior programmer, team leader, project leader etc)
5. **Communication** phase is used to understand business problem as well as it tries to identify all type of problems and gather all type of information.
6. **Planning** is essential because it divides work into manageable different parts and distributed among the various teams.
7. **Modeling** will be accomplished by different teams simultaneously. It includes various phases - Business modelling (BM), Data modeling (DM), Process modeling (PM), Application Generation (AG) and Testing & Turnover (TT).

- *Business modeling* collects all essential information about the product from market view or from the business point of view such as - what information drives the business process?, what information is generated and how is it done and where does it go and all such questions.
- *Data modeling* shows the flow of data using some techniques like DFD.
- *Process modeling* performs each process which handles the flow of data. It decides the activities performed in each process.

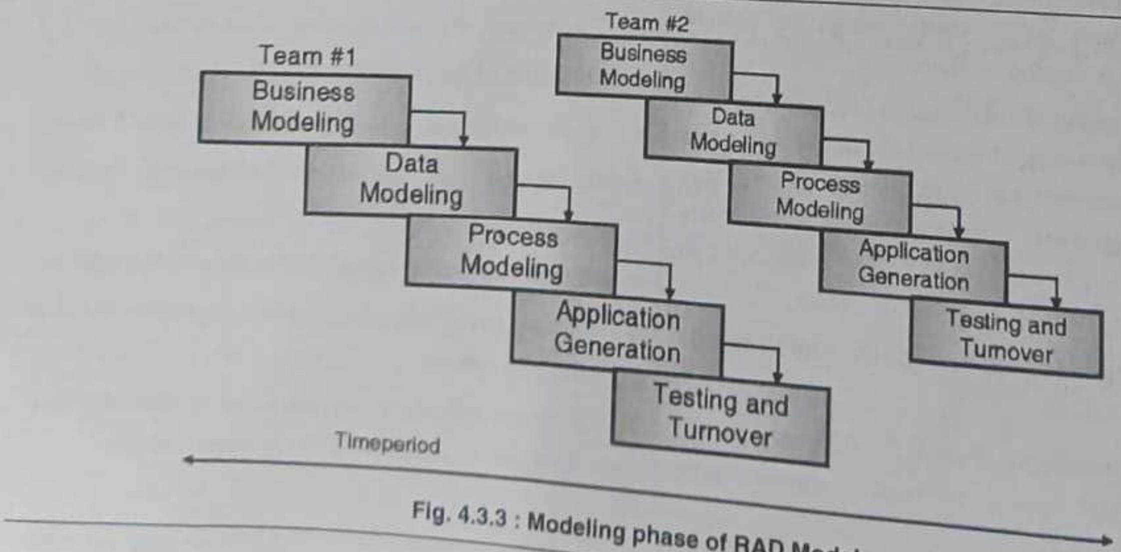

Fig. 4.3.3 : Modeling phase of RAD Model

- *Application Generation* is about using a set of automated tools to facilitate the construction of the software say for example, the 4th GL techniques.
- *Testing and Turn over* : Many of the components that will be used in the development of the proposed system might have already been tested since RAD focuses on reuse which reduces overall testing time. It is needed to concentrate on testing the new components.
- Once modeling activity of each team is over, it immediately starts construction phase.

8. In **construction** phase, each team develops the code of our product as well as performs testing. It reuses old software components and develops new functions which are called as automatic code generation.
9. **Deployment** collects each code from different team and clubs them into single project implement it and if required then perform iteration among the previous phases.

This RAD model is illustrated in Fig. 4.3.4

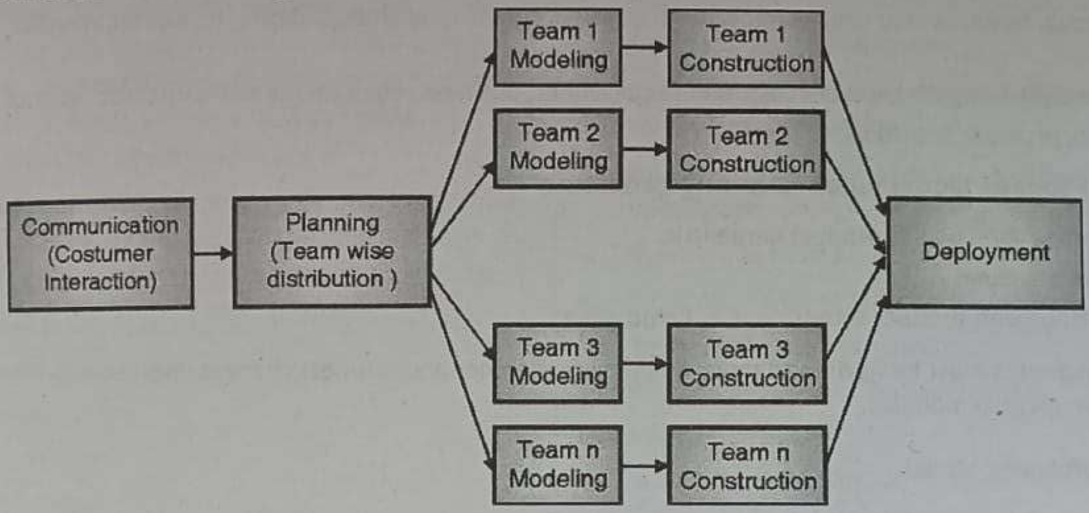

Fig. 4.3.4 : RAD model

Advantages

1. It requires very less time to develop the product.
2. Planning and Design is performed before construction so it is not lengthy.
3. Product may be produced before its delivery date.

Disadvantages

1. Large projects require sufficient or more human resources.
2. If developers and customers do not interact with each other then whole project may elapse.
3. If system is to be modularized properly then RAD is problematic.
4. If high performance is the Issue then RAD does not work.
5. It also doesn't work in high technical risk.

4.4 Evolutionary Process Models

- The *Evolutionary Process model* removes the limitations of the *Incremental process model.*
- Therefore, first we will list out the drawbacks of incremental model that forced to design the evolutionary process model.

The main drawbacks of Incremental process model

1. Overall design of the system is not so good.
2. It occupies large space in memory.
3. Some time speed of software is also slow.
4. Delivery date of the product is not decide.

CHAPTER 5 UNIT I

Agile Software Development

Syllabus

Agile software development : Agile methods, Plan-driven and agile development, Extreme programming, Agile project management, Scaling agile methods.

5.1 Introduction

Agility

- Agility is the unending process, which always accepts specific requirement of the product from the customers or end user then it checks whether the requirements are growth oriented or not ?
- If it is growth oriented then it aggressively changes the existing system.
- Dissatisfaction with the conventional SDLC models led to the creation of agile methods.

The agile methods focus on

- The code rather than design
- An iterative software development approach
- Deliver working product quickly and evolves quickly to meet changing requirements.
- **Main Objective of agile methods** : To reduce overhead in the software process and respond quickly to changing requirements without excessive rework.

5.1.1 The 12 Principles of Agile A Q.11

- The Agile alliance [AG103] defines 12 principles to achieve agility

1. **Customer involvement :** Customer satisfaction by continuous delivery of the software.
2. **Accept changes :** Always accept changes in requirement though it is late in development.
3. **Incremental delivery :** Deliver working software frequently in shorter time span.
4. **People not process :** Business people and developers must work together during complete development of the project.
5. Develop the project from the software project team. It includes :
   - Motivate each individual of the project.
   - Provide then necessary resources and environment to build the project.
   - Trust them to get the job done.
6. Do the **face to face conversation,** if essential or urgent.
7. Primary measure of the progress is the working software.
8. Agile process promotes sustainable development. (That means somebody from agile team doing a good task to helpful for development should get promotions).
9. Keeps Continues track to technical excellence and best design increases agility.
10. It collects information about amount of work not done and tries to find out reasons of that.
11. Agile team collects requirement, and fulfill them using best architectures.
12. Continuous thinking about how the performance of team can be increased.

Agility can be applied to any software process.

5.2 Agile Methodology

Agile methodologies address the following 3 key problems of software project.

1. It is difficult to predict which requirement is changing and which is not changing?
2. Some type of software, design and construction phase are related with each other, so again it is difficult to predict exactly how much time required to design process.

Application of XP

1. Planning game
2. Small releases
3. Customer acceptance tests
4. Simple design
5. Pair-driving programming
6. Test driving development
7. Refactoring the program / code
8. Continuous progressive integration
9. Collective code ownership
10. Use of coding standards
11. Metaphor for modeling
12. Maintaining sustainable pace of development.

5.5 Agile Project Management A Q.12

- Agile Project Management is a latest project management strategy mainly applied to project management practice in software development.
- With the advancement of software development process and technologies used, the traditional SDLC models such as waterfall, incremental, RAD are no more robust enough to fulfill the business requirements.
- More flexible SDLC models are required so as to address the agility of the requirements and therefore came up the 'Agile' software development approach.
- Since agile development model is different from conventional models, so agile project management is used for project management practice.

Scope of Agile Project Management

- In an agile project management, the entire software development team is self-managed or say, self-organized and it is not just the project manager's responsibility.
- The project manager is not assumed as the 'boss' of the team. He is there only to facilitate and coordinate the activities and resources required for quality and faster software development.
- The complete team uses their common sense to take decisions and this makes sure that there is no delay in making management decision therefore they can progress faster.
- The agile project management function also demonstrates the leadership skills in motivating each other which helps retaining the spirit among the team members.

Responsibilities of an agile project manager

- Maintains the agile practices in the project team.
- Facilitates and coordinates the activities
- Encourages effective and open communication within the team.
- Helps team members to turn the product backlog into working software functionality.
- Facilitates the enhanced resources (tools and practices) required for quality and faster software development

Agile project management does not

- Manage the software development team.
- Dominate the team members and take decisions.
- Order the team members to perform tasks or routines.
- Allot tasks to the team members.

5.6 Scaling Agile Methods

- Agile methods are suitable for small and medium sized projects that developed by a small co-located team.
- One of the main advantage of such small co-located team is improved communication which is possible when everyone is working together.
- Scaling up agile methods is about changing enhancing or changing the features of agile methods such that it becomes suitable for larger projects having multiple development teams located at different locations.
- But while scaling agile methods it is essential to maintain agile fundamental principles - flexible planning, frequent releases, continuous integration, test-driven development and good team communications.

Scaling out and scaling-up

- 'Scaling up' is using agile methods for developing large software systems having multiple development teams.
- 'Scaling out' is introducing agile methods across a large organization with many years of software development experience.

Scaling up to large systems

- For large systems development, it is not possible to just focus on the code you also need to focus on design and documentation.
- Cross-team communication must happen through regular phone and video conferences, short electronic meetings where teams update each other on progress.
- Continuous integration i.e. maintain frequent system builds and regular releases of the system.

Scaling out to large companies

- Project managers who do not have experience of agile methods may not accept the risk of using a new approach.
- Large organizations often have quality procedures and standards that all projects are expected to follow and, because of their bureaucratic nature, these are likely to be incompatible with agile methods.
- There may be some restrictions implemented on using agile methods, especially in those organizations that have a long history of using conventional models.

Review Questions

Q. 1 What is agility and what does the agile methods focus on ?

Q. 2 State the fundamental principles of agile methodology.

Q. 3 Explain the difference between plan-driven and agile development approach.

Q. 4 Write short notes on FDD and DSDM.

Q. 5 Explain the Scrum approach and state the need of sprints.

Q. 6 Explain XP process model in detail.

Q. 7 Explain pair-programming.

Q. 8 What is the advantage of test driven development.

Q. 9 What are the responsibilities of an Agile Project Manager ?

Q. 10 What is meant by Scaling agile methods. Explain in detail

❑❑❑