

**SY B.Sc. IT Sem. 4**

<b>1.</b>	<b>Attempt the following:</b>
a.	Define software engineering. Explain the Software Development Life Cycle (SDLC) steps in brief.
Ans:	<p><b>Software Engineering definition (1M)</b></p> <p>The software development life cycle (SDLC) is a framework defining tasks performed at each step in the software development process. The life cycle defines a methodology for improving the quality of software and the overall development process.</p> <p>There are following six phases in every Software development life cycle model: (1M)</p> <ol style="list-style-type: none"><li>1. Requirement gathering and analysis</li><li>2. Design</li><li>3. Implementation or coding</li><li>4. Testing</li><li>5. Deployment</li><li>6. Maintenance</li></ol> <p><b>Explanation (3M)</b></p> <p><b>1) Requirement gathering and analysis:</b></p> <p>Business requirements are gathered in this phase. Meetings with managers, stake holders and users are held in order to determine the requirements like; Who is going to use the system? How will they use the system? What data should be input into the system? What data should be output by the system? After requirement gathering these requirements are analyzed for their validity. Requirement Specification document is created which serves the purpose of guideline for the next phase of the model.</p> <p><b>2) Design:</b></p> <p>In this phase the system and software design is prepared from the requirement specifications which were studied in the first phase. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture. The system design specifications serve as input for the next phase of the model.</p> <p><b>3) Implementation / Coding:</b></p> <p>On receiving system design documents, the work is divided in modules/units and actual coding is started. Since, in this phase the code is produced so it is the main focus for the developer. This is the longest phase of the software development life cycle.</p> <p><b>4) Testing:</b></p> <p>After the code is developed it is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase. During this phase all types of functional testing like unit testing, integration testing, system testing, acceptance testing are done as well as non-functional testing are also done.</p> <p><b>5) Deployment:</b></p> <p>After successful testing the product is deployed to the customer for their use. As soon as the product is given to the customers they will first do the beta testing. If any changes are required or if any bugs are caught, then they will report it to the engineering team. Once changes are made or the bugs are fixed then the final deployment will happen.</p> <p><b>6) Maintenance:</b></p> <p>Once when the customers starts using the developed system then the actual problems comes up and needs to be solved from time to time. This process where the care is taken for the developed product is known as maintenance</p>
b.	Explain the classification of the software requirements?
Ans:	Software system requirements are often classified as functional requirements or non- functional requirements:

**1. Functional requirements:** These are statements of services the system should provide, how the system should react to particular inputs, and how the system should behave in particular situations. In some cases, the functional requirements may also explicitly state what the system should not do.

**Example of functional requirements are :-** a) Descriptions of data to be entered into the system b) Descriptions of operations performed by each screen c) Who can enter the data into the system d) Descriptions of system reports or other outputs

**2. 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 constraints imposed by standards. Nonfunctional requirements often apply to the system as a whole, rather than individual system features or services.

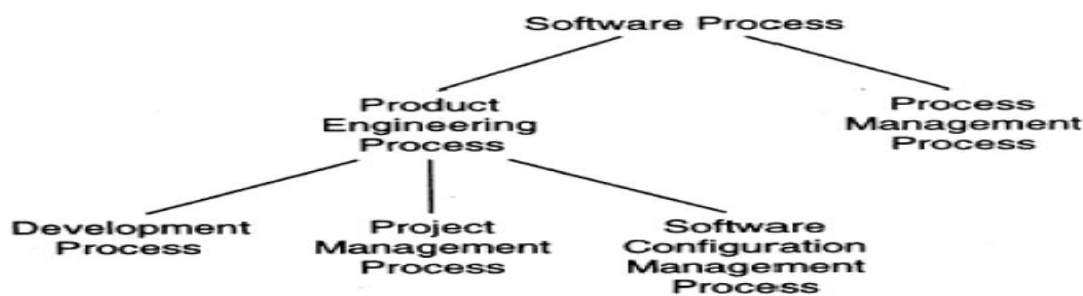
**i) Product requirements:-** These requirements specify or constrain the behaviour of the Software Examples include performance requirements on how fast the system must execute and how much memory it requires, reliability requirements that set out the acceptable failure rate, security requirements, and usability requirements.

**ii) Organizational requirements:-** These requirements are broad system requirement derived from policies and procedures in the customer's and developer's organization. Examples include operational process requirements that define how the system will be used, development process requirements that specify the programming language, the development environment or process standards to be used.

**iii) External requirements:-** All requirements that are derived from factors external to the system and its development process. These may include regulatory requirements that set out what must be done for the system to be approved for use by a regulator, legislative requirements that must be followed to ensure that the system operates within the law; and ethical requirements that ensure that the system will be acceptable to its users and the general public.

c. What are the components of software process? Explain.

Ans: Components of Software Process:  
The processes that deal with the technical and management issues of software development are collectively called the software process.  
The relationship between these major component processes is shown in Figure



There are clearly two major components in a software process:-

**A development process:** The development process specifies all the engineering activities that need to be performed.

**A project management process:** The management process specifies how to plan and control these activities so that cost, schedule, quality, and other objectives are met.

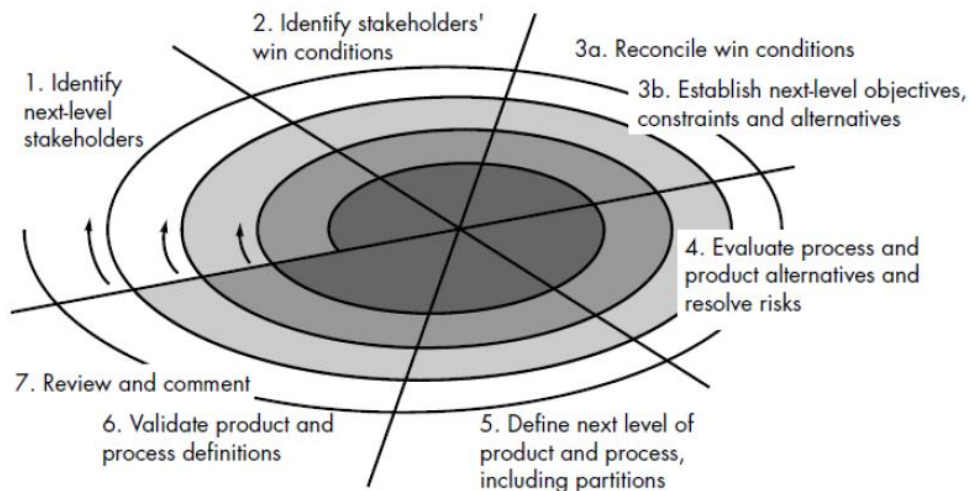
Development and project management processes are the key to achieving the objectives of delivering the desired software satisfying the user needs, while ensuring high productivity and quality. During the project many products are produced which are typically composed of many items. These items keep evolving 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 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).												
d.	Explain the structure of software requirement document.												
Ans:	<p>Structure of Software Requirement Document is as follows:</p> <table border="1"> <tr> <td>Preface</td><td>This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.</td></tr> <tr> <td>Introduction</td><td>This should describe the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.</td></tr> <tr> <td>Glossary</td><td>This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.</td></tr> <tr> <td>User requirements definition</td><td>Here, you describe the services provided for the user. The non-functional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.</td></tr> <tr> <td>System architecture</td><td>This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.</td></tr> <tr> <td>System requirements specification</td><td>This should describe the functional and non-functional requirements in more detail. If necessary, further detail may also be added to the non-functional requirements. Interfaces to other systems may be defined.</td></tr> </table>	Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.	Introduction	This should describe the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.	Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.	User requirements definition	Here, you describe the services provided for the user. The non-functional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.	System architecture	This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.	System requirements specification	This should describe the functional and non-functional requirements in more detail. If necessary, further detail may also be added to the non-functional requirements. Interfaces to other systems may be defined.
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.												
Introduction	This should describe the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.												
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.												
User requirements definition	Here, you describe the services provided for the user. The non-functional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.												
System architecture	This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.												
System requirements specification	This should describe the functional and non-functional requirements in more detail. If necessary, further detail may also be added to the non-functional requirements. Interfaces to other systems may be defined.												
e.	Write short note on spiral model.												
Ans:	<p>The spiral model, originally proposed by Boehm, it is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model. It provides the potential for rapid development of incremental versions of the software. Using the spiral model, software is developed in a series of incremental releases. During early iterations, the incremental release might be a paper model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced. A spiral model is divided into a number of framework activities, also called task regions. spiral model that contains six task regions:</p> <ul style="list-style-type: none"> <li>• <b>Customer communication</b>—tasks required to establish effective communication between developer and customer.</li> <li>• <b>Planning</b>—tasks required to define resources, timelines, and other project related information.</li> <li>• <b>Risk analysis</b>—tasks required to assess both technical and management risks.</li> <li>• <b>Engineering</b>—tasks required to build one or more representations of the application.</li> <li>• <b>Construction and release</b>—tasks required to construct, test, install, and provide user support</li> <li>• <b>Customer evaluation</b>—tasks required to obtain customer feedback based on evaluation of the software representations created during the engineering stage and implemented during the installation stage.</li> </ul> <p>Each of the regions is populated by a set of work tasks, called a task set.</p>												

## SPIRAL MODEL



f. What are the principles of agile method?

Ans: The principles of agile method are as follows:

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

2. **Attempt the following:**

a. State and explain the emergent systems properties with example.

Ans: **Emergent System Properties:** It has properties that are properties of the system as a whole. The emergent properties cannot be attributed to any specific part of the system. Rather, they only emerge once the system components have been integrated.

There are two types of emergent properties:

1. **Functional emergent properties** when the purpose of a system only emerges after its components are integrated. For example, a bicycle has the functional property of being a transportation device once it has been assembled from its components.

2. **Non-functional emergent properties**, which relate to the behavior of the system in its operational environment. Reliability, performance, safety, and security are examples of emergent properties. These are critical for computer based systems, as failure to achieve a minimum defined level in these properties usually makes the system unusable. Some users may not need some of the system functions



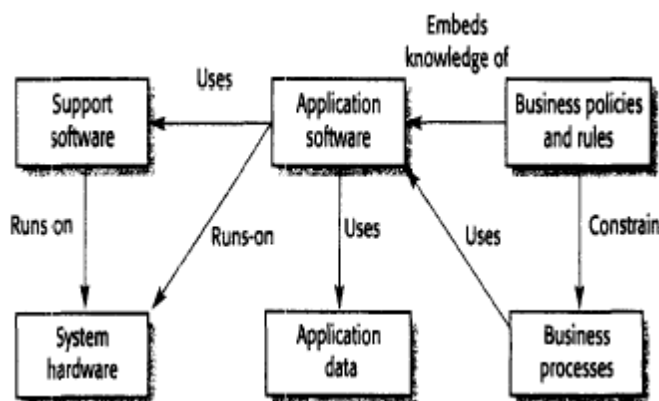
,so the system may be acceptable without them. However, a system that is unreliable or too slow is likely to be rejected by all its users.

Property	Description
Volume	The volume of a system (the total space occupied) varies depending on how the component assemblies are arranged and connected.
Reliability	System reliability depends on component reliability but unexpected interactions can cause new types of failures and therefore affect the reliability of the system.
Security	The security of the system (its ability to resist attack) is a complex property that cannot be easily measured. Attacks may be devised that were not anticipated by the system designers and so may defeat built-in safeguards.
Repairability	This property reflects how easy it is to fix a problem with the system once it has been discovered. It depends on being able to diagnose the problem, access the components that are faulty, and modify or replace these components.
Usability	This property reflects how easy it is to use the system. It depends on the technical system components, its operators, and its operating environment.

b. Explain the legacy system with the help of diagram.

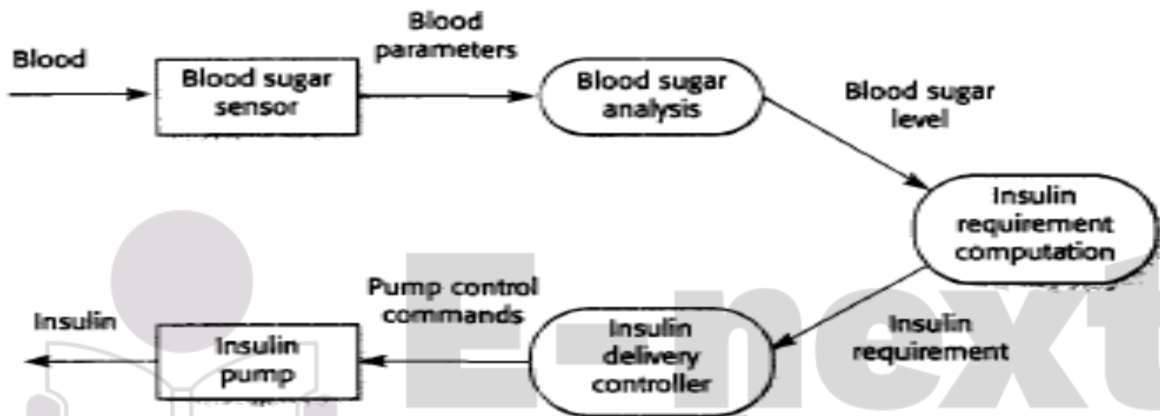
Ans: **Legacy Systems:**

- Legacy system are Socio-technical computer based systems that have been developed in the past using an obsolete technology.
- These system includes various sub-systems, processes and procedures which are working on older ways and rely on legacy systems that are difficult to change.
- These systems are crucial to the operation of a business and it is often too risky to be discarded such as bank customer accounting system and aircraft maintenance system etc.
- Legacy systems constraints new business processes and consumes a high proportion of company budgets. Various issues are:- 1) Do we throw away and restart or continue to maintain ? 2) What are the economics (cost and risk) of each approach. 3) If system depends on other COTS, will upgrades to those be available?

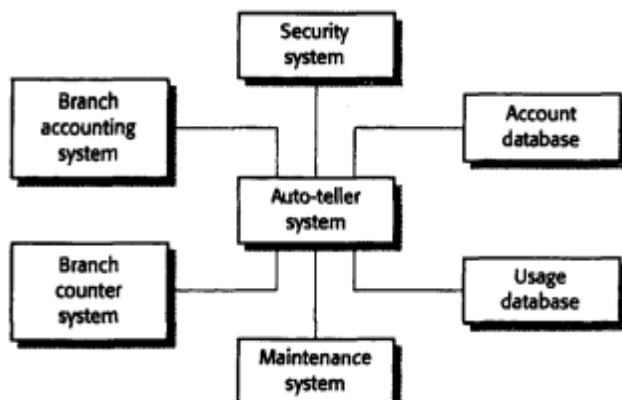


#### LEGACY SYSTEM COMPONENTS

- **System hardware** – Legacy systems have been written for mainframe hardware that is no longer available, that is expensive to maintain and that may not be compatible with current organizational IT purchasing policies
- **Support software** – The legacy system may rely on a range of support software from the operating system and utilities provided by the hardware manufacturer through to the compilers used for system development
- **Application software** – The application system that provides the business services is usually composed of a number of separate programs that have been developed at different times.
- **Application Data** – These are the data that are processed by the application system. In many legacy systems, an immense volume of data has accumulated over the lifetime of the system. This data may be

	<p>inconsistent and may be duplicated in several files</p> <ul style="list-style-type: none"> <li>• <b>Business process</b> – These are processes that are used in the business to achieve some business objective. Business processes may be designed around a legacy system and constraint by the functionality that it provides</li> <li>• <b>Business policies and rules</b> – These are definitions of how the business should be carried out and constraints on the business</li> </ul>
c.	Explain the simple critical system with example.
Ans:	<p><b>Critical Systems:</b></p> <ul style="list-style-type: none"> <li>• Systems failure that can result insignificant economic losses, physical damage or threats to human life.</li> <li>• They are technical or socio technical systems that people depend on</li> <li>• If these systems fail to deliver their services as expected then serious problems and significant loses may result critical systems</li> <li>• <b>Safety-critical system</b> – System whose failure may result in injury, loss of life or serious environmental damage. Example Control system for a chemical manufacturing plant</li> <li>• <b>Mission-critical system</b> – System whose failure may result in the failure of some goal directed activity. Example Navigational system for a spacecraft</li> <li>• <b>Business-critical system</b> – System whose failure may result in very high costs for the business using that system. Example Customer accounting system in a bank.</li> </ul>  <p>The system is an insulin system as a safety-critical system. A software controlled insulin pump is used by diabetics to simulate the function of the pancreas which manufactures insulin , an essential hormone that metabolises blood glucose. It measures blood glucose(sugar) using a micro-sensor and computes the insulin dose required to metabolize the glucose. A common condition where the human pancreas is unable to produce sufficient quantities of a hormone is called insulin.</p>
d.	Explain the importance of feasibility study in requirements engineering process.
Ans:	<p><b>Feasibility Study:</b> For all new systems the requirement engineering process should start with feasibility study.</p> <p>Feasibility study An estimate is made of whether the identified user needs may be satisfied using current software and hardware technologies. The study considers whether the proposed system will be cost-effective from a business point of view and if it can be developed within existing budgetary constraints. A feasibility study should be relatively cheap and quick. The result should inform the decision of whether or not to go ahead with a more detailed analysis.</p> <ul style="list-style-type: none"> <li>• The input to the feasibility study is preliminary requirements, an outline description of how the system is intended to support business process.</li> <li>• The results is the report that recommends whether or not it is worth carrying on with the project.</li> <li>• Thus a feasibility study is a short focused study that aims to answer a number of questions like system contribution, system implementation and system integration.</li> <li>• If the system does not support the business objectives it has no real value to the business.</li> <li>• Carrying out a feasibility study involves information assessment, information collection and report writing.</li> <li>• The information assessment phase identifies the information that is required and once the information is gathered discussion with various sources can be done.</li> </ul>
e.	Write short note on i) Context model, ii) Object model.

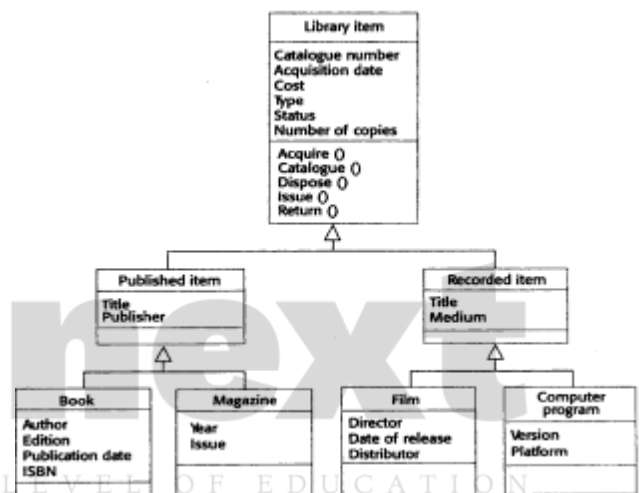
Ans: **Context Model:** • At an early stage in the requirements elicitation and analysis process boundaries of the system must be decided involving system stakeholders In some cases the boundary between a system and its environment is relatively clear.



**For example** where an automated system is replacing an existing manual or computerized system the environment of the new system is usually the same as the existing system where as in other cases the stakeholders decide the boundary. For example in the library system the user decides the boundary whether to include library catalogues for accessing or not. Once some decisions on the boundaries of the system has been made part of the activity is definition of that context.

**Object Model:** • Expressing the system requirements using object model, designing using objects and developing using languages like C++ and Java

Object models developed during requirements analysis are used to represent both data and its process. They combine some uses of dataflow and semantic models • They are also useful for showing how entities in the system may be classified and composed of other entities. Objects are executable entities with attributes and services of the object class and many objects can be created from a class • The following diagram shows an object class in UML as a vertically oriented rectangle with three sections – name of the object, class attributes, operations associated with the object.



f. Explain requirement validation process checks on the requirements in the requirement document.

Ans: **Requirements Validation:** It is concerned with showing that the requirements actually define the system that the customer wants. It overlaps analysis in that it is concerned with finding problems with the requirements. It is important because errors in requirements documentation can lead to extensive rework costs when they are discovered during development process. The cost of fixing requirements problems is much greater than repairing design or coding errors. **Various checks carried out on requirements in requirements document are –**

**Validity checks** – A system is needed to perform certain functions

**Consistency checks** – Requirements should not be contradictory or of the same system function

**Completeness checks** – Requirements document should include all functions and constraints **Realism**

**checks** – Requirements should be checked to ensure that they could actually be implemented

**Verifiability** – Requirements should always be written so that they are verifiable

### Requirements Validation Technique

• **Requirements reviews** – The requirements are analysed systematically by a team of reviewers

• **Prototyping** – In this approach to validation an executable model of the system is demonstrated to end users and customers

• **Test case generation** – Tests for the requirements are devised as a part of validation process. Developing tests from the user requirements before any code is written in an integral part of extreme programming

• **Requirement review** is a manual process that involves people from both client and contractor organizations. • They check the requirements document for irregularities and errors. • Requirements reviews can be informal (involves contractors discussing requirements with stake holders) and formal

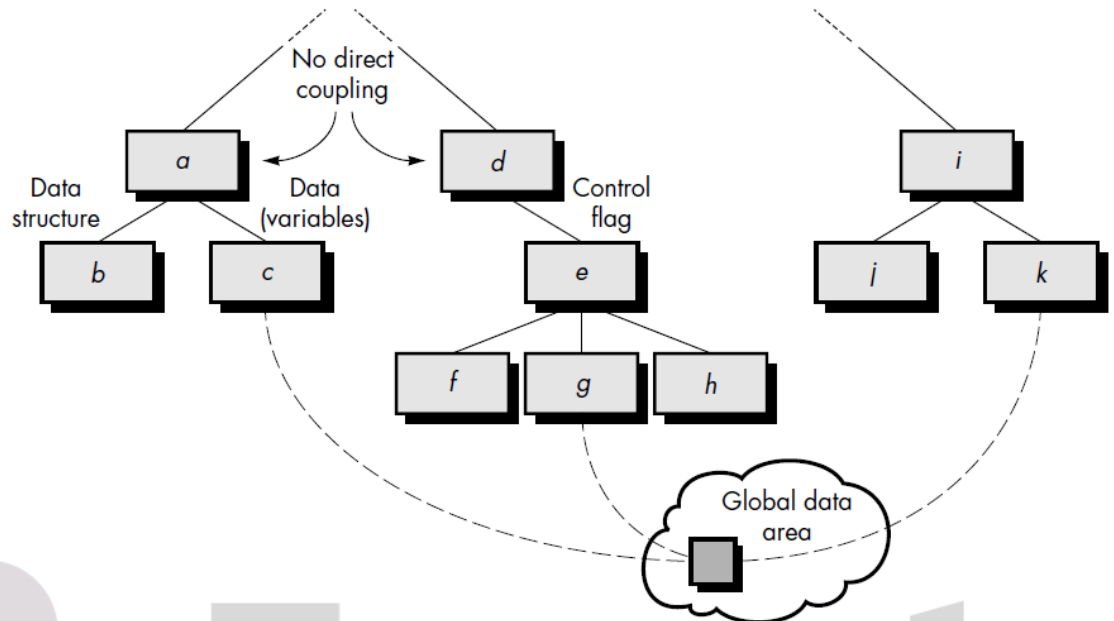
	(the development team walks through the system requirements) • Reviewers may also check for – Verifiability – Comprehensibility – Traceability – Adaptability
<b>3.</b>	<b>Attempt the following:</b>
a.	Write short note on architectural design decisions.
Ans:	<p><b>Architectural design</b> is a creative process where you design a system organization that will satisfy the functional and non-functional requirements of a system. Because it is a creative process, the activities within the process depend on the type of system being developed, the background and experience of the system architect, and the specific requirements for the system. It is therefore useful to think of architectural design as a series of decisions to be made rather than a sequence of activities. During the architectural design process, system architects have to make a number of structural decisions that profoundly affect the system and its development process.</p> <p>Based on their knowledge and experience, they have to consider the following fundamental questions about the system:</p> <ol style="list-style-type: none"> <li>1. Is there a generic application architecture that can act as a template for the system that is being designed?</li> <li>2. How will the system be distributed across a number of cores or processors?</li> <li>3. What architectural patterns or styles might be used?</li> <li>4. What will be the fundamental approach used to structure the system?</li> <li>5. How will the structural components in the system be decomposed into subcomponents?</li> <li>6. What strategy will be used to control the operation of the components in the system?</li> <li>7. What architectural organization is best for delivering the non-functional requirements of the system?</li> <li>8. How will the architectural design be evaluated?</li> <li>9. How should the architecture of the system be documented?</li> </ol> <p>Although each software system is unique, systems in the same application domain often have similar architectures that reflect the fundamental concepts of the domain. For example, application product lines are applications that are built around a core architecture with variants that satisfy specific customer requirements. When designing a system architecture, you have to decide what your system and broader application classes have in common, and decide how much knowledge from these application architectures you can reuse.</p>
b.	Write short note on modular decomposition styles.
Ans:	<p><b>Modularity:</b> Software is divided into separately named and addressable components, often called modules, that are integrated to satisfy problem requirements.</p> <p><b>Modular decomposability.</b> If a design method provides a systematic mechanism for decomposing the problem into subproblems, it will reduce the complexity of the overall problem, thereby achieving an effective modular solution.</p> <p><b>Modular Decomposition</b></p> <p>A module is a well-defined component of a software system. A module is part of a system that provides a set of services to other modules</p> <p><b>Properties of a decomposition:</b> <input type="checkbox"/> Cohesion <input type="checkbox"/> Coupling <input type="checkbox"/> Complexity <input type="checkbox"/> Correctness <input type="checkbox"/> Correspondence <input type="checkbox"/> Strategies for decomposition <input type="checkbox"/> Information Hiding <input type="checkbox"/> Layering</p> <p><b>Cohesion &amp; Coupling</b></p> <ol style="list-style-type: none"> <li>1. <b>Cohesion:</b> The reason that elements are found together in a module. <p>A cohesive module performs a single task within a software procedure, requiring little interaction with procedures being performed in other parts of a program.</p> <p>A module that performs tasks that are related logically is <b>logically cohesive</b>. When a module contains tasks that are related by the fact that all must be executed with the same span of time, the module exhibits <b>temporal cohesion</b>. When all processing elements concentrate on one area of a data structure, <b>communicational cohesion</b> is present. High cohesion is characterized by a module that performs one distinct procedural task. As we have already noted, it is unnecessary to determine the precise level of cohesion. Rather it is important to strive for high cohesion and recognize low cohesion so that software design can be modified to achieve greater functional independence.</p> </li> <li>2. <b>Coupling:</b> Strength of interconnection between modules</li> </ol>



Coupling is a measure of interconnection among modules in a software structure. Coupling depends on the interface complexity between modules, the point at which entry or reference is made to a module, and what data pass across the interface. The highest degree of coupling, content coupling, occurs when one module makes use of data or control information maintained within the boundary of another module. Secondly, content coupling occurs when branches are made into the middle of a module. This mode of coupling can and should be avoided.

**FIGURE 13.6**

Types of coupling

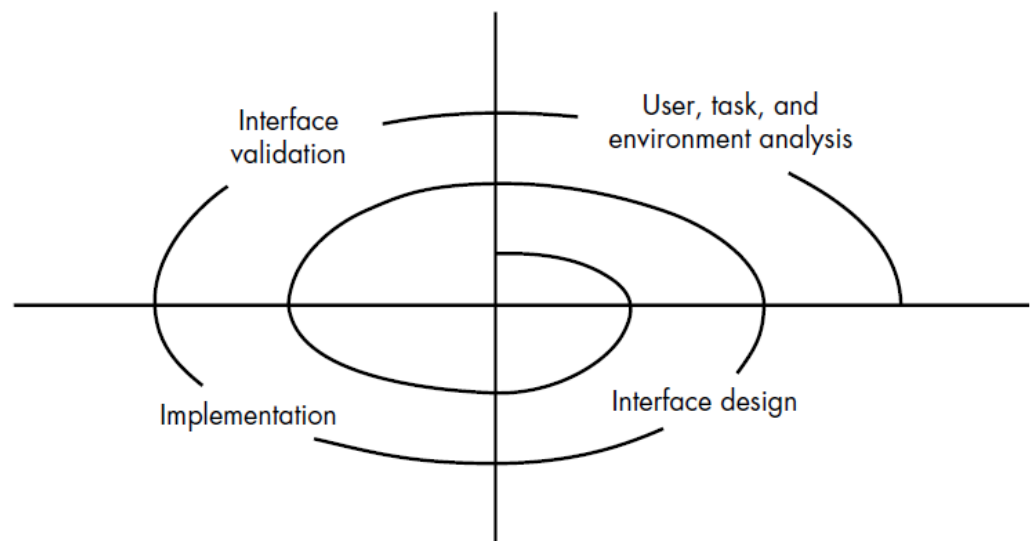


c. Explain user interface design process with the help of diagram.

**User interface design:** The process of designing the way in which system users can access system functionality, and the way that information produced by the system is displayed. User interface design creates an effective communication medium between a human and a computer. Following a set of interface design principles, design identifies interface objects and actions and then creates a screen layout that forms the basis for a user interface prototype. The overall **process for designing a user interface** begins with the creation of different models of system function. The human- and computer-oriented tasks that are required to achieve system function are then delineated; design issues that apply to all interface designs are considered; tools are used to prototype and ultimately implement the design model; and the result is evaluated for quality.

**FIGURE 15.1**

The user interface design process

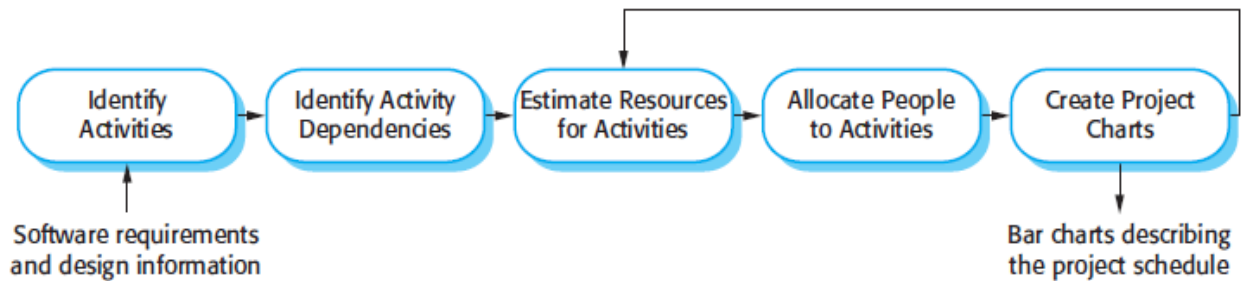


**The User Interface Design Process** The design process for user interfaces is iterative and can be represented using a spiral model user interface design process encompasses four distinct framework activities [MAN97]:

1. User, task, and environment analysis and modeling
2. Interface design

	<p>3. Interface construction</p> <p>4. Interface validation</p> <p>Each of these tasks will occur more than once, with each pass around the spiral representing additional elaboration of requirements and the resultant design. In most cases, the implementation activity involves prototyping—the only practical way to validate what has been designed.</p> <p>Once general requirements have been defined, a more detailed task analysis is conducted. Those tasks that the user performs to accomplish the goals of the system are identified, described, and elaborated. As we have already noted, the activities described in this section occur iteratively. Therefore, there is no need to attempt to specify every detail (for the analysis or design model) on the first pass. Subsequent passes through the process elaborate task detail, design information, and the operational features of the interface.</p> <p><b>For example,</b> a user interface designer may propose a graphical UI for an embedded system that requires a great deal of processing and so overloads the processor in the system.</p>
d.	Explain the risk management process.
Ans:	<p>Risk management is one of the most important jobs for a project manager. Risk management involves anticipating risks that might affect the project schedule or the quality of the software being developed (Hall, 1998; Ould, 1999). You can think of a risk as something that you'd prefer not to have happen. Risks may threaten the project, the software that is being developed, or the organization.</p> <p><b>There are, therefore, three related categories of risk:</b></p> <ol style="list-style-type: none"> <li>1. <b>Project risks</b> Risks that affect the project schedule or resources.</li> <li>2. <b>Product risks</b> Risks that affect the quality or performance of the software being developed.</li> <li>3. <b>Business risks</b> Risks that affect the organization developing or procuring the software.</li> </ol> <p>An outline of the <b>process of risk management</b> is illustrated in Figure 1. It involves several stages:</p> <ol style="list-style-type: none"> <li>1. <b>Risk identification</b> You should identify possible project, product, and business risks.</li> <li>2. <b>Risk analysis</b> You should assess the likelihood and consequences of these risks.</li> <li>3. <b>Risk planning</b> You should make plans to address the risk, either by avoiding it or minimizing its effects on the project.</li> <li>4. <b>Risk monitoring</b> You should regularly assess the risk and your plans.</li> </ol> <pre> graph LR     A([Risk Identification]) --&gt; B([Risk Analysis])     B --&gt; C([Risk Planning])     C --&gt; D([Risk Monitoring])     D --&gt; A     A --&gt; E[List of Potential Risks]     B --&gt; F[Prioritized Risk List]     C --&gt; G[Risk Avoidance and Contingency Plans]     D --&gt; H[Risk Assessment]   </pre>
e.	Write short note on project scheduling.
Ans:	<p><b>Project scheduling</b> is the process of deciding how the work in a project will be organized as separate tasks, and when and how these tasks will be executed. You estimate the calendar time needed to complete each task, the effort required, and who will work on the tasks that have been identified. You also have to estimate the resources needed to complete each task, such as the disk space required on a server, the time required on specialized hardware,</p> <p>The initial schedule is used to plan how people will be allocated to projects and to check the progress of the project against its contractual commitments. In traditional development processes, the complete schedule is initially developed and then modified as the project progresses. In agile processes, there has to be an overall schedule that identifies when the major phases of the project will be completed.</p> <p>An iterative approach to scheduling is then used to plan each phase. Scheduling in plan-driven projects (Figure ) involves breaking down the total work involved in a project into separate tasks and estimating the time required to complete each task. Tasks should normally last at least a week, and no longer than 2 months.. Some of these tasks are carried out in parallel, with different people working on different components of the system.</p> <p>If the project being scheduled is similar to a previous project, previous estimates may be reused.</p>

However, projects may use different design methods and implementation languages, so experience from previous projects may not be applicable in the planning of a new project. A good rule of thumb is to estimate as if nothing will go wrong, then increase your estimate to cover anticipated problems. A further contingency factor to cover unanticipated problems may also be added to the estimate. This extra contingency factor depends on the type of project, the process parameters (deadline, standards, etc.), and the quality and experience of the software engineers working on the project.



f. What is quality assurance? What are the quality standards types? Explain.

**Quality Assurance{QA}:** The terms ‘quality assurance’ and ‘quality control’ are widely used in manufacturing industry. **Quality assurance (QA)** is the definition of processes and standards that should lead to high-quality products and the introduction of quality processes into the manufacturing process. Quality control is the application of these quality processes to weed out products that are not of the required level of quality.

In the software industry, different companies and industry sectors interpret quality assurance and quality control in different ways. Sometimes, quality assurance simply means the definition of procedures, processes, and standards that are aimed at ensuring that software quality is achieved.

In other cases, quality assurance also includes all configuration management, verification, and validation activities that are applied after a product has been handed over by a development team.

So ‘**quality assurance**’ to include verification and validation and the processes of checking that quality procedures have been properly applied. The QA team in most companies is responsible for managing the release testing process. Companies are responsible for checking that the system tests provide coverage of the requirements and that proper records are maintained of the testing process.

**Software standards:** play a very important role in software quality management. The quality assurance is the definition or selection of standards that should apply to the software development process or software product. As part of this QA process, tools and methods to support the use of these standards may also be chosen. Once standards have been selected for use, project-specific processes have to be defined to monitor the use of the standards and check that they have been followed. Software standards are important for three reasons:

1. Standards capture wisdom that is of value to the organization.
2. Standards provide a framework for defining what ‘quality’ means in a particular setting.
3. Standards assist continuity when work carried out by one person is taken up and continued by another.

There are **two related types** of software engineering standard that may be defined and used in software quality management: .

**1.Product standards** These apply to the software product being developed. They include document standards, such as the structure of requirements documents, documentation standards, such as a standard comment header for an object class definition, and coding standards, which define how a programming language should be used.

**2. Process standards** These define the processes that should be followed during software development. They should encapsulate good development practice. Process standards may include definitions of specification, design and validation processes, process support tools, and a description of the documents that should be written during these processes.

Product standards	Process standards
Design review form	Design review conduct
Requirements document structure	Submission of new code for system building
Method header format	Version release process
Java programming style	Project plan approval process
Project plan format	Change control process
Change request form	Test recording process

**4. Attempt the following:**

a. Define verification and validation. Explain software inspection in v & v process.

Ans: **Verification** involves checking that the software conforms to its specification.

**Validation**, however, is a more general process. The aim of validation is to ensure that the software system meets the customer's expectations.

Verification and validation are not the same thing, although they are often confused.

'**Validation:** Are we building the right product?'

'**Verification:** Are we building the product right?'

These definitions tell us that the role of verification involves checking that the software conforms to its specification. Validation, however, is a more general process. The aim of validation is to ensure that the software system meets the customer's expectations.

*Software inspections or peer reviews* analyse and check system representations such as the requirements document, design diagrams and the program source code. You can use inspections at all stages of the process. Inspections may be supplemented by some automatic analysis of the source text of a system or associated documents. Software inspections and automated analyses are static V & V techniques, as you don't need to run the software on a computer. Software Inspections

**Software inspection is a static V and V process** in which a software system is reviewed to find errors, omissions and anomalies. Generally, inspections focus on source code, but any readable representation of the software such as its requirements or a design model can be inspected. When you inspect a system, you use knowledge of the system, its application domain and the programming language or design model to discover errors.

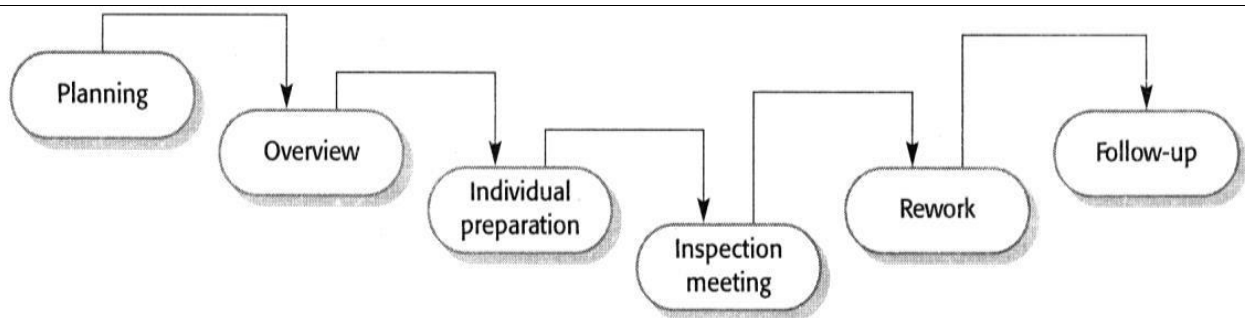
There are three major advantages of inspection over testing:

1. During testing, errors can mask (hide) other errors. Once one error is discovered, you can never be sure if other output anomalies are due to a new error or are side effects of the original error. Because inspection is a static process, you don't have to be concerned with interactions between errors. Consequently, a single inspection session can discover many errors in a system.

2. Incomplete versions of a system can be inspected without additional costs. If a program is incomplete, then you need to develop specialised test harnesses to test the parts that are available. This obviously adds to the system development costs.

3. As well as searching for program defects, an inspection can also consider broader quality attributes of a program such as compliance with standards, portability and maintainability. You can look for inefficiencies, inappropriate algorithms and poor programming style that could make the system difficult to maintain and update.

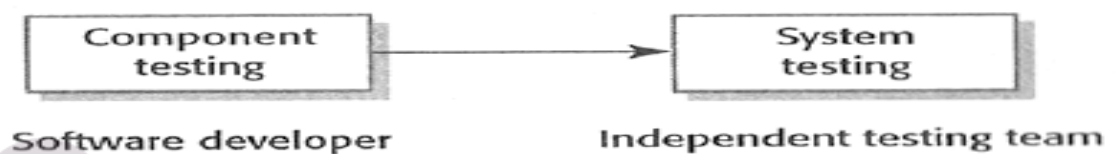




System overview presented to inspection team • Code and associated documents are distributed to inspection team in advance • Inspection takes place and discovered errors are noted • Modifications are made to repair discovered errors • Re-inspection may or may not be required

b. Write short note on component testing.

Ans: This model of the testing process is appropriate for large system development—but for smaller systems, or for systems that are developed through scripting or reuse there are often fewer distinct stages in the process. A more abstract view of software testing is shown in Figure 18. The two fundamental testing activities are component testing—testing the parts of the system—and system testing—testing the system as a whole.

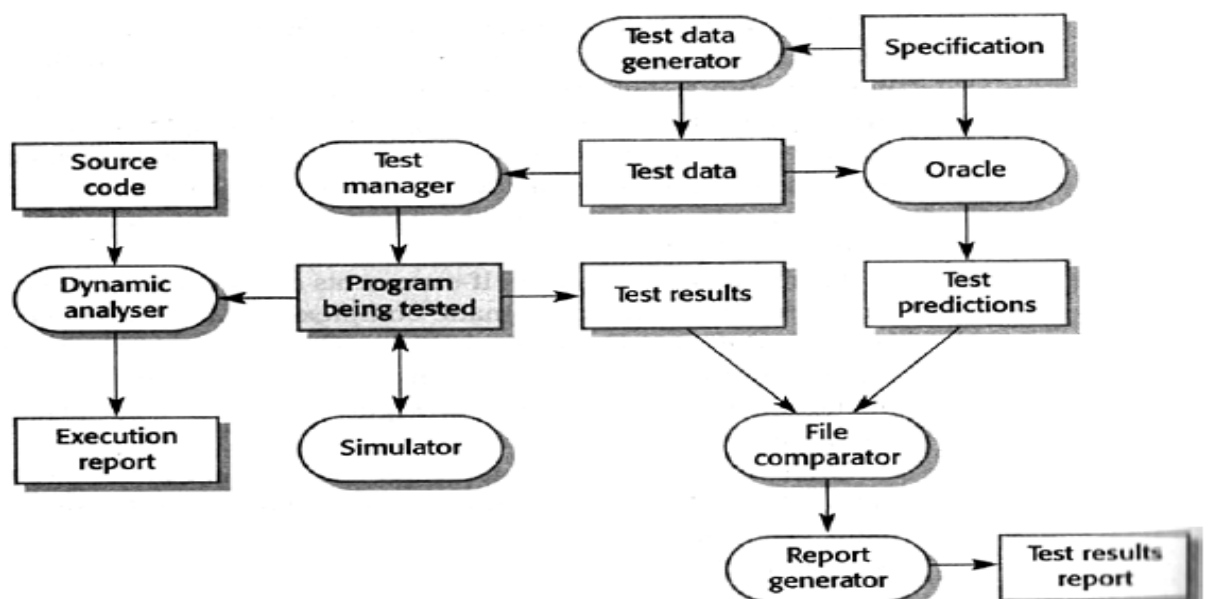


**Fig. 18 : Testing phases**

The aim of the component testing stage is to discover defects by testing individual program components. These components may be functions, objects or reusable components. During system testing, these components are integrated to form subsystems or the complete system. At this stage system testing should focus on establishing that the system meets its functional and non-functional requirements, and does not behave in unexpected ways. Inevitably, defects in components that have been missed during earlier testing are discovered during system testing.

c. Explain the test automation.

Ans: Test Automation Testing is an expensive and laborious phase of the software process. As a result, testing tools were among the first software tools to be developed. These tools now offer a range of facilities and their use can significantly reduce the costs of testing.



**Fig. 33 : A testing workbench**

	One approach to test automation where a testing framework such as JUnit is used for regression testing. JUnit is a set of Java classes that the user extends to create an automated testing environment. Each individual test is implemented as an object and a test runner runs all of the tests. The tests themselves should be written in such a way that they indicate whether the tested system has behaved as expected.
d.	Write short note on function point(FP) and Line of Code(LOC) measures.
Ans:	<p>There are two types of metric that have been used:</p> <p><b>1. Size-related metrics.</b> These are related to the size of some output from an activity. The most commonly used size related metric is <b>lines of delivered source code</b>. Other metrics that may be used are the number of delivered object code instructions or the number of pages of system documentation.</p> <p><b>2.Function-related metrics.</b> These are related to the overall functionality of the delivered software. Productivity is expressed in terms of the amount of useful functionality produced in some given time. <b>Function points</b> and object points are the best-known metrics of this type.</p> <p><b>Lines of source code per programmer-month</b> (LOC/pm) is a widely used software productivity metric. You can compute LOC/pm by counting the total number of lines of source code that are delivered, then divide the count by the total time in programmer months required to complete the project. This time therefore includes the time required for all other activities (requirements, design, coding, testing and documentation) involved in software development.</p> <p><b>Line of Code(LOC):</b>Direct measures of the software engineering process include cost and effort applied. Direct measures of the product include lines of code (LOC) produced, execution speed, memory size, and defects reported over some set period of time. Indirect measures of the product include functionality, quality, complexity, efficiency, reliability, maintainability, and many other "-abilities"</p> <p><b>Function-oriented metrics:</b> were first proposed by Albrecht [ALB79], who suggested a measure called the function point. Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments of software complexity.</p>
e.	Explain the Cost Constructive Model (COCOMO) with the formula for computing duration of project and manpower efforts for project.
Ans:	<p><b>The COCOMO Model</b> A number of algorithmic models have been proposed as the basis for estimating the effort, schedule and costs of a software project. These are conceptually similar but use different parameter values. The model discussed here is the COCOMO model. The COCOMO model is an empirical model that was derived by collecting data from a large number of software projects. These data were analysed to discover formulae that were the best fit to the observations. These formulae link the size of the system and product, project and team factors to the effort to develop the system.</p> <p>COCOMO II is a well-documented and nonproprietary estimation model. COCOMO II was developed from earlier COCOMO cost estimation models, which were largely based on original code development (Boehm, 1981; Boehm and Royce, 1989). The COCOMO II model takes into account more modern approaches to software development, such as rapid development using dynamic languages, development by component composition, and use of database programming.</p> <p><b>Project Duration and Staffing:</b> Estimate how long the software will take to develop and when staff will be needed to work on the project. The development time for the project is called the <b>project schedule</b>. Increasingly, organisations are demanding shorter development schedules so that their products can be brought to market before their competitor's.</p> <p>The relationship between the number of staff working on a project, the total effort required and the development time is not linear. As the number of staff increases, more effort may be needed. The reason for this is that people spend more time communicating and defining interfaces between the parts of the system developed by other people. Doubling the number of staff (for example) therefore does not mean that the duration of the project will be halved.</p> <p>The COCOMO model includes a formula to estimate the calendar time (TDEV) required to complete a project. The time computation formula is the same for all COCOMO levels :</p> $TDEV = 3 \times (PM)^{(0.33 + 0.2 \cdot (B-1.01))}$ <p>PM is the effort computation and B is the exponent computed, as discussed above (B is 1 for the early</p>

prototyping model). This computation predicts the nominal schedule for the project. However, the predicted project schedule and the schedule required by the project plan are not necessarily the same thing. The planned schedule may be shorter or longer than the nominal predicted schedule. and the COCOMO II model predicts this :

$$TDEV = 3 \times (PM)^{(0.33 + 0.2^*(B - 1.01))} \times SCEDPercentage / 100$$

SCEDPercentage is the percentage increase or decrease in the nominal schedule.

To illustrate the COCOMO development schedule computation, assume that 60 months of effort are estimated to develop a software system . From the schedule equation, the time required to complete the

project is:  $TDEV = 3 \times (60)^{0.36} = 13 \text{ months}$

f. Explain the software cost estimation technique.

#### Estimation Techniques:

Organisations need to make software effort and cost estimates. To do so, one or more of the techniques described in Figure 1 may be used. All of these techniques rely on experience-based judgements by project managers who use their knowledge of previous projects to arrive at an estimate of the resources required for the project. However, there may be important differences between past and future projects. Some examples of the changes that may affect estimates based on experience include:

1. Distributed object systems rather than mainframe-based systems
2. Use of web services
3. Use of ERP or database-centred systems
4. Use of off-the-shelf software rather than original system development
5. Development for and with reuse rather than new development of all parts of a system
6. Development using scripting languages such as TCL or Perl (Ousterhout, 1998) 7.

The use of CASE tools and program generators rather than unsupported software development.

Technique	Description
Algorithmic cost modelling	A model is developed using historical cost information that relates some software metric (usually its size) to the project cost. An estimate is made of that metric and the model predicts the effort required.
Expert judgement	Several experts on the proposed software development techniques and the application domain are consulted. They each estimate the project cost. These estimates are compared and discussed. The estimation process iterates until an agreed estimate is reached.
Estimation by analogy	This technique is applicable when other projects in the same application domain have been completed. The cost of a new project is estimated by analogy with these completed projects. Myers (Myers, 1989) gives a very clear description of this approach
Parkinson's Law	Parkinson's Law states that work expands to fill the time available. The cost is determined by available resources rather than by objective assessment. If the software has to be delivered in 12 months and 5 people are available, the effort required is estimated to be 60 person-months
Pricing to win	The software cost is estimated to be whatever the customer has available to spend on the project. The estimated effort depends on the customer's budget and not on the software functionality

5. Attempt the following:

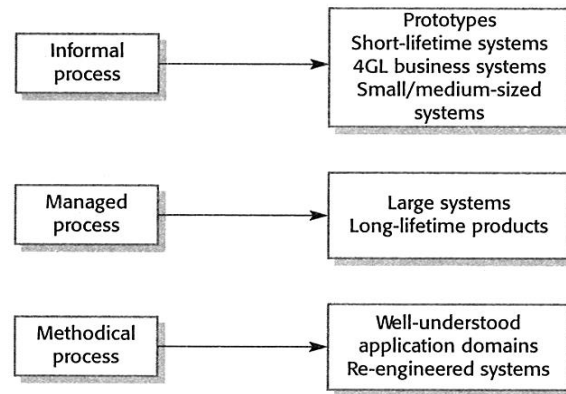
a. Describe the classification of process.

Ans: **Process Classification**

- 1) **Informal processes.** When there is no strictly defined process model, the development team chooses the process that they will use. Informal processes may use formal procedures such as configuration management, but the procedures and the relationships between procedures are defined

as required by the development team.

- 2) **Managed processes.** A defined process model is used to drive the development process. The process model defines the procedures, their scheduling and the relationships between the procedures.
- 3) **Methodical processes.** When some defined development method or methods (such as systematic methods for object-oriented design) are used, these processes benefit from CASE tool support for design and analysis processes.
- 4) **Improving processes.** Processes that have inherent improvement objectives have a specific budget for improvements and procedures for introducing such improvements. As part of this, quantitative process measurement may be introduced.



**Fig. 8 : Process applicability**

These classifications obviously overlap, and a process may fall into several classes. For example, the process may be informal in that it is chosen by the development team. The team may choose to use a particular design method. They may also have a process-improvement capability. In this case, the process would be classified as informal, methodical and improving.

b. Explain the CMMI process improvement framework.

**Ans: The CMMI Process Improvement Framework**  
The CMMI model (Ahem, et al., 2001) is intended to be a framework for process improvement that has broad applicability across a range of companies. Its staged version is compatible with the Software CMM and allows an organisation's system development and management processes to be assessed and assigned a maturity level from 1 to 5. Its continuous version allows for a finer-grain classification and rates 24 process areas on a scale from 1 to 6.

Category	Process area
Process management	Organisational process definition Organisational process focus Organisational training Organisational process performance Organisational innovation and deployment
Project management	Project planning Project monitoring and control Supplier agreement management Integrated project management Risk management Integrated teaming Quantitative project management
Engineering	Requirements management Requirements development Technical solution Product integration Verification



	Validation
Support	Configuration management Process and product quality management Measurement and analysis Decision analysis and resolution Organisational environment for integration Causal analysis and resolution

**Fig. 10 :** Process areas in the CMMI

The model is very complex (more than 1,000 pages of description), so I have radically simplified it for discussion here:

- 1) *Process areas.* The CMMI identifies 24 process areas that are relevant to software process capability and improvement. These are organised into four groups in the continuous CMMI model. These groups and related process areas are listed in Figure 10.
- 2) *Goals.* Goals are abstract descriptions of a desirable state that should be attained by an organisation. The CMMI has specific goals that are associated with each process area and that define the desirable state for that area. It also defines generic goals that are associated with the institutionalisation of good practice.
- 3) *Practices.* Practices in the CMMI are descriptions of ways to achieving a goal. Up to seven specific and generic practices may be associated with each goal within each process area. Examples of recommended practices are shown in Figure 12. However, the CMMI recognises that it is the goal rather than the way that goal is reached that is important. Organisations may use any appropriate practices to achieve any of the CMMI goals—they do not have to take the CMMI recommendations.

Goal	Process Area
Corrective actions are managed to closure when the project's performance or results deviate significantly from the plan	Specific goal in project monitoring and control
Actual performance and progress of the project is monitored against the project plan	Specific goal in project monitoring and control
The requirements are analysed and validated, and a definition of the required functionality is developed	Specific goal in requirements development
Root causes of defects and other problems are systematically determined	Specific goal in causal analysis and resolution
The process is institutionalised as a defined process	Generic goal

c. Explain the services as a reusable components.

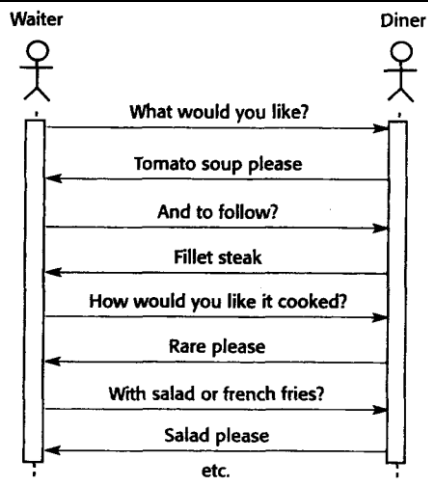
**Ans: Services as Reusable Components**

Software systems are constructed by composing software components that are based on some standard component model. Services are a natural development of software components where the component model is, in essence, the set of standards associated with web services. A service can therefore be defined as:

A loosely coupled, reusable software component that encapsulates discrete functionality, which may be distributed and programmatically accessed. A web service is a service that is accessed using standard Internet and XML-based protocols.

A critical distinction between a service and a software component as defined in CBSE is that services should be independent and loosely coupled. That is, they should always operate in the same way, irrespective of their execution environment. Their interface is a provides' interface that provides access to the service functionality. Services are intended to be independent and usable in different contexts. Therefore, they do not have a requires' interface that, in CBSE, defines the other system components that must be present.

Services may also be distributed over the Internet. They communicate by exchanging messages, expressed in XML, and these messages are distributed using standard Internet transport protocols such as HTTP and TCP/IP. A service defines what it needs from another service by setting out its requirements in a message and sending it to that service. The receiving service parses the message, carries out the computation and, on completion, sends a message to the requesting service. This service then parses the reply to extract the required information. Unlike software components, services do not 'call' methods associated with other services.



**Fig. :** Synchronous interaction when ordering a meal.

To illustrate the difference between communication using method calls and communication using message passing, consider a situation where you are ordering a meal in a restaurant. When you have a conversation with the waiter, you are involved in a series of synchronous interactions that define your order. This is comparable to components interacting in a software system, where one component calls methods from other components. The waiter writes down your order along with the order of the other people with you, then passes this message, which includes details of everything that has been ordered, to the kitchen to prepare the food. Essentially, the waiter service is passing a message to the kitchen service defining the food to be prepared.

Figure 15, which shows the synchronous ordering process, and in Figure 16, which shows a hypothetical XML message, which is self-explanatory, that defines an order made by the table of three people. The difference is clear—the waiter takes the order as a series of interactions, with each interaction defining part of the order. However, the waiter has a single interaction with the kitchen where the message passed defines the complete order.

```

<starter>
  <dish name = "soup" type = "tomato" />
  <dish name = "soup" type = "fish" />
  <dish name = "pigeon salad" />
</starter>
<main course>
  <dish name = "steak" type = "sirloin" cooking = "medium" />
  <dish name = "steak" type = "fillet" cooking = "rare" />
  <dish name = "sea bass">
</main>
<accompaniment>
  <dish name = "french fries" portions = "2" />
  <dish name = "salad" portions = "1" />
</accompaniment>
  
```

**Fig.:** A restaurant order expressed as an XML message.

When you intend to use a web service, you need to know where the service is located (its URI) and the details of its interface. These are described in a service description expressed in an XML-based language called WSDL (Web Service Description Language). The WSDL specification defines three things about a Web service. It defines *what* the service does, how it communicates and where to find it:

- 1) The 'what' part of a WSDL document, called an interface, specifies what operations the service supports, and defines the format of the messages that are sent and received by the service.
- 2) The 'how' part of a WSDL document, called a binding, maps the abstract interface to a concrete set of protocols. The binding specifies the technical details of how to communicate with a Web service.
- 3) The 'where' part of a WSDL document, called (confusingly) a service, describes where to locate a specific Web service implementation.

d. Explain the application framework.

Ans: **Application frameworks**

	<p>Frameworks provide support for generic features that are likely to be used in all applications of a similar type.</p> <p>For example, a user interface framework will provide support for interface event handling and will include a set of widgets that can be used to construct displays. For example, in a user interface framework, the developer defines display layouts that are appropriate to the application being implemented.</p> <p>Fayad and Schmidt (1997) discuss three classes of frameworks:</p> <ol style="list-style-type: none"> <li>1. System infrastructure frameworks</li> <li>2. Middleware integration frameworks</li> <li>3. Enterprise application frameworks</li> </ol> <p>These are concerned with specific application domains such as telecommunications or financial systems (Baumer, et al., 1997). These embed application domain knowledge and support the development of end-user applications.</p> <p>Web application frameworks (WAFs) are a more recent and very important type of framework. WAFs that support the construction of dynamic websites are now widely available. Web application frameworks usually incorporate one or more specialized frameworks that support specific application features. Although each framework includes slightly different functionality,</p> <p>Application frameworks and software product lines obviously have much in common. They both support a common architecture and components, and require new development to create a specific version of a system. The main differences between these approaches are as follows:</p> <ol style="list-style-type: none"> <li>1. Application frameworks rely on object-oriented features such as inheritance and polymorphism to implement extensions to the framework. Generally, the framework code is not modified and the possible modifications are limited to whatever is allowed by the framework. Software product lines are not necessarily created using an object-oriented approach. Application components are changed, deleted, or rewritten. There are no limits, in principle at least, to the changes that can be made.</li> <li>2. Application frameworks are primarily focused on providing technical rather than domain-specific support. For example, there are application frameworks to create web-based applications. A software product line usually embeds detailed domain and platform information. For example, there could be a software product line concerned with web-based applications for health record management.</li> <li>3. Software product lines are often control applications for equipment. For example, there may be a software product line for a family of printers. This means that the product line has to provide support for hardware interfacing. Application frameworks are usually software-oriented and they rarely provide support for hardware interfacing.</li> <li>4. Software product lines are made up of a family of related applications, owned by the same organization. When you create a new application, your starting point is often the closest member of the application family, not the generic core application.</li> </ol>
e.	Write short note on commercial-off-the-shelf(COTS) product reuse.
Ans:	<p><b>COTS product reuse:</b></p> <p>A commercial-off-the-shelf (COTS) product is a software system that can be adapted to the needs of different customers without changing the source code of the system. Virtually all desktop software and a wide variety of server products are COTS software. Because this software is designed for general use, it usually includes many features and functions. It therefore has the potential to be reused in different environments and as part of different applications. Torchiano and Morisio (2004) also discovered that using open source products were often used as COTS products. That is, the open source systems were used without change and without looking at the source code. COTS products are adapted by using built-in configuration mechanisms that allow the functionality of the system to be tailored to specific customer needs.</p> <p>There are two types of COTS product reuse, namely COTS-solution systems and COTS-integrated systems. COTS-solution systems consist of a generic application from a single vendor that is configured to customer requirements. COTS-integrated systems involve integrating two or more COTS systems (perhaps from different vendors) to create an application system.</p>

COTS-solution systems	COTS-integrated systems
Single product that provides the functionality required by a customer	Several heterogeneous system products are integrated to provide customized functionality
Based around a generic solution and standardized processes	Flexible solutions may be developed for customer processes
Development focus is on system configuration	Development focus is on system integration
System vendor is responsible for maintenance	System owner is responsible for maintenance
System vendor provides the platform for the system	System owner provides the platform for the system

f. What are the architectural patterns for distributed systems? Explain Master-Slave architecture.

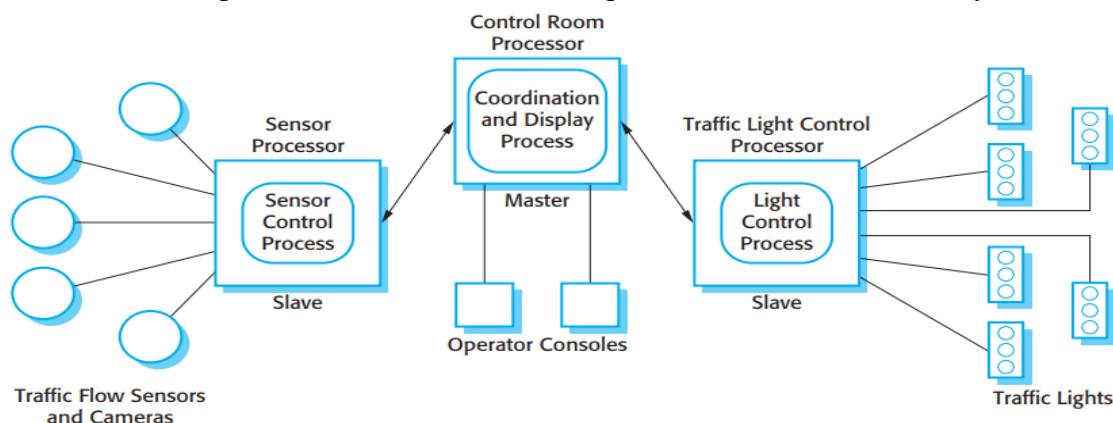
Ans: **Architectural patterns for distributed systems**

Five architectural styles:

1. **Master-slave architecture**, which is used in real-time systems in which guaranteed interaction response times are required.
2. **Two-tier client-server architecture**, which is used for simple client-server systems, and in situations where it is important to centralize the system for security reasons. In such cases, communication between the client and server is normally encrypted.
3. **Multitier client-server architecture**, which is used when there is a high volume of transactions to be processed by the server.
4. **Distributed component architecture**, which is used when resources from different systems and databases need to be combined, or as an implementation model for multi-tier client-server systems.
5. **Peer-to-peer architecture**, which is used when clients exchange locally stored information and the role of the server is to introduce clients to each other. It may also be used when a large number of independent computations may have to be made.

#### Master-slave architectures

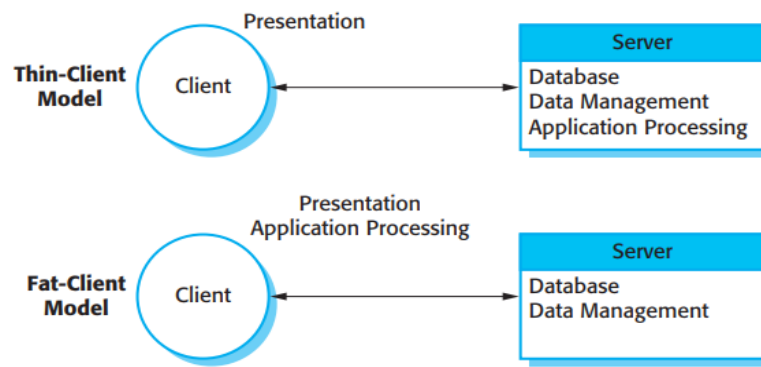
Master-slave architectures for distributed systems are commonly used in realtime systems where there may be separate processors associated with data acquisition from the system's environment, data processing, and computation and actuator management. Actuators, are devices controlled by the software system that act to change the system's environment. For example, an actuator may control a valve and change its state from 'open' to 'closed'. The 'master' process is usually responsible for computation, coordination, and communications and it controls the 'slave' processes. 'Slave' processes are dedicated to specific actions, such as the acquisition of data from an array of sensors.



A set of distributed sensors collects information on the traffic flow. The sensor control process polls the sensors periodically to capture the traffic flow information and collates this information for further processing. The sensor processor is itself polled periodically for information by the master process that is concerned with displaying traffic status to operators, computing traffic light sequences and accepting operator commands to modify these sequences. The control room system sends commands to a traffic light control process that converts these into signals to control the traffic light hardware. The master control room system is itself organized as a client-server system, with the client processes running on



the operator's consoles.



**Figure 18.8** Thin- and fat-client architectural models



# E-next

THE NEXT LEVEL OF EDUCATION