

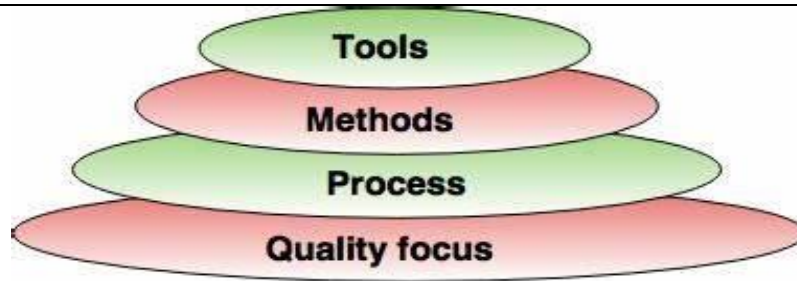
<b>St. Peter's Engineering College (Autonomous)</b> <b>Dullapally (P), Medchal, Hyderabad – 500100.</b> <b>QUESTION BANK WITH ANSWERS</b>				Dept.	:	CSM
				Academic Year 2024-25		
Subject Code	:	AS22-66ES01	Subject	:	SOFTWARE ENGINEERING	
Class/Section	:	B. Tech.	Year	:	II	Semester : I
Faculty	:	Mrs.D.Divya (B & D sec) , Mr.G.Pavan (A& E Sec) , Mrs.M.Benita Roy (C Sec)				

BLOOMS LEVEL					
Remember	L1	Understand	L2	Apply	L3
Analyze	L4	Evaluate	L5	Create	L6

Q. No	Question (s)	Marks	BL	CO
<b>UNIT – I</b>				
1(a)	Define the software engineering.	1M	L1	C215.1
	Ans: It is a systematic, quantifiable and disciplined approach to operate and maintain a software.			
1(b)	Mention the steps involved in software development life cycle	1M	L1	C215.1
	Ans. The steps followed to develop a software is called as process in Software engineering .The steps are : 1)Software Requirements 2)Analysis 3)Design 4)Implementation 5)Testing 6)Deployment 7)Maintenance.			
1(c)	Define product in the evolving nature of software.	1M	L1	C215.1
	Ans. The software which produces, manages and displays information that software is called as product.			
1(d)	Define system software, application software.	1M	L1	C215.1
	Ans. 1) System Software: The software which provide service to the other software's is known as System Software. Ex: Operating System. 2) Application Software: The software which provides services to the end user known as Application Software. Ex : E-Mail app.			
1(e)	Define software myth.	1M	L1	C215.1
	Ans. Beliefs about software and the process used to built it can be traced to the earliest days of computing myths have a number of attributes that have made them insidious.			
2(a)	Mention the characteristics of software.	3M	L1	C215.1
	Ans. Characteristics of Software: → Software is developed or engineered but not manufactured → Software doesn't wearout → Software is custom built not user built.			
2(b)	List the frame work activities, umbrella activities.	3M	L1	C215.1
	Ans. -> <b>Frame work activities:</b> 1)communication 2)Planning 3)Modeling 4)Construction 5)Deployment -> <b>Umbrella activities:</b> 1)Software project tracking and control 2)Risk management 3)Software quality assurance 4)Formal technical reviews 5)Measurement 6)Software configuration management 7)Reusability management 8)Work product preparation and production			

2(c)	Mention the advantages of waterfall model.	3M	L1	C215.1
	Ans. Waterfall model – Advantages → Simple and easy to understand and use → Easy to manage → Works well for smaller projects → Clearly defined stages → Well - undertood milestones			
2(d)	Mention the advantages of spiral model.	3M	L1	C215.1
	Ans. Spiral model – Advantages → Suitable for large scale product → Development is fast → Efficient cost estimation → Proper risk management → Involves customer feedback			
2(e)	List the phases of agile model.	3M	L1	C215.1
	Ans. Phases of Agile model 1)Requirement gathering 2)Design the requirements 3)Construction / iteration 4)Testing / quality assurance 5)Deployment 6)Feedback			
3(a)	Explain regarding the evolving nature of software, changing nature of software.	5M	L2	C215.1
	<p><b>Ans</b> Software plays a dual role. Those are            1)Product 2)Vehicle</p> <p><b><u>PRODUCT:-</u></b>            The software which produces, manages, displays information that is called product</p> <p><b><u>VEHICLE:-</u></b>            The software which is able to transmit or process the simple data multimedia information from one place to another place is called as vehicle</p> <p>1)Software can control the products            2)software can create new software            3)software can provide different functionalities for different users.</p> <p><b><u>CHANGING NATURE OF SOFTWARE:-</u></b>            There are 7 types of software those are:-            1)system software            2)Applications software            3)engineering &amp; scientific software            4)embedded software            5)product line software            6)website software            7)AI software            1)<u>System software:</u></p>			

	<p>System software is a collection of programs which are written to service other programs. Some system software processes complex but determinate, information structures. Other system application process largely indeterminate data. Sometimes when, the system software area is characterized by the heavy interaction with computer hardware that requires scheduling, resource sharing, and sophisticated process management.</p> <p>Ex:-OS</p> <p>2) <u>Application software:</u> Application software is defined as programs that solve a specific business need. Application in this area process business or technical data in a way that facilitates business operation or management technical decision making. In addition to convention data processing application, application software is used to control business function in real time.</p> <p>Ex:-email</p> <p>3. <u>Engineering and Scientific Software:</u> This software is used to facilitate the engineering function and task. however modern application within the engineering and scientific area are moving away from the conventional numerical algorithms. Computer-aided design, system simulation, and other interactive applications have begun to take a real- time and even system software characteristic.</p> <p>Ex:-CAD, Unigraphics &amp; cations</p> <p>4. <u>Embedded Software:</u> Embedded software resides within the system or product and is used to implement and control feature and function for the end-user and for the system itself. Embedded software can perform the limited and esoteric function or provided significant function and control capability.</p> <p>Ex:- Micro oven, geysers, heaters</p> <p>5. <u>Product-line Software:</u> Designed to provide a specific capability for use by many different customers, product line software can focus on the limited and esoteric marketplace or address the mass consumer market.</p> <p>Ex:- cloud computing in data base management</p> <p>6. <u>Web Application:</u> It is a client-server computer program which the client runs on the web browser. In their simplest form, Web apps can be little more than a set of linked hypertext files that present information using text and limited graphics. However, as e-commerce and B2B application grow in importance. Web apps are evolving into a sophisticate computing environment that not only provides a standalone feature, computing function, and content to the end user.</p> <p>Ex:-SPEC website</p> <p>7. <u>Artificial Intelligence Software:</u> Artificial intelligence software makes use of a nonnumerical algorithm to solve a complex problem that is not amenable to computation or straightforward analysis. Application within this area includes robotics, expert system, pattern recognition, artificial neural network, theorem proving and game playing.</p> <p>Ex:-PUBG</p>			
3(b)	Explain how the Software Engineering is considered as a Layered Technology.	5M	L2	C215.1
Ans	<ul style="list-style-type: none"> <li>• Software engineering is a fully layered technology</li> <li>• To develop a software, we need to go from one layer to another.</li> </ul>			



**Fig. - Software Engineering Layers**

• All these layers are related to each other and each layer demands the fulfillment of the previous layer. The layered technology consists of:

1. Quality focus :The characteristics of good quality software are:

- Correctness of the functions required to be performed by the software.
- Maintainability of the software
- Integrity i.e. providing security so that the unauthorized user cannot access information or data.
- Usability i.e. the efforts required to use or operate the software.

2. Process:

- It is the base layer or foundation layer for the software engineering.
- The software process is the key to keep all levels together.
- It defines a framework that includes different activities and tasks.
- In short, it covers all activities, actions and tasks required to be carried out for software development.

3. Methods:

- The method provides the answers of all 'how-to' that are asked during the process.
- It provides the technical way to implement the software.
- It includes collection of tasks starting from communication, requirement analysis, analysis and design modelling, program construction, testing and support.

4. Tools :

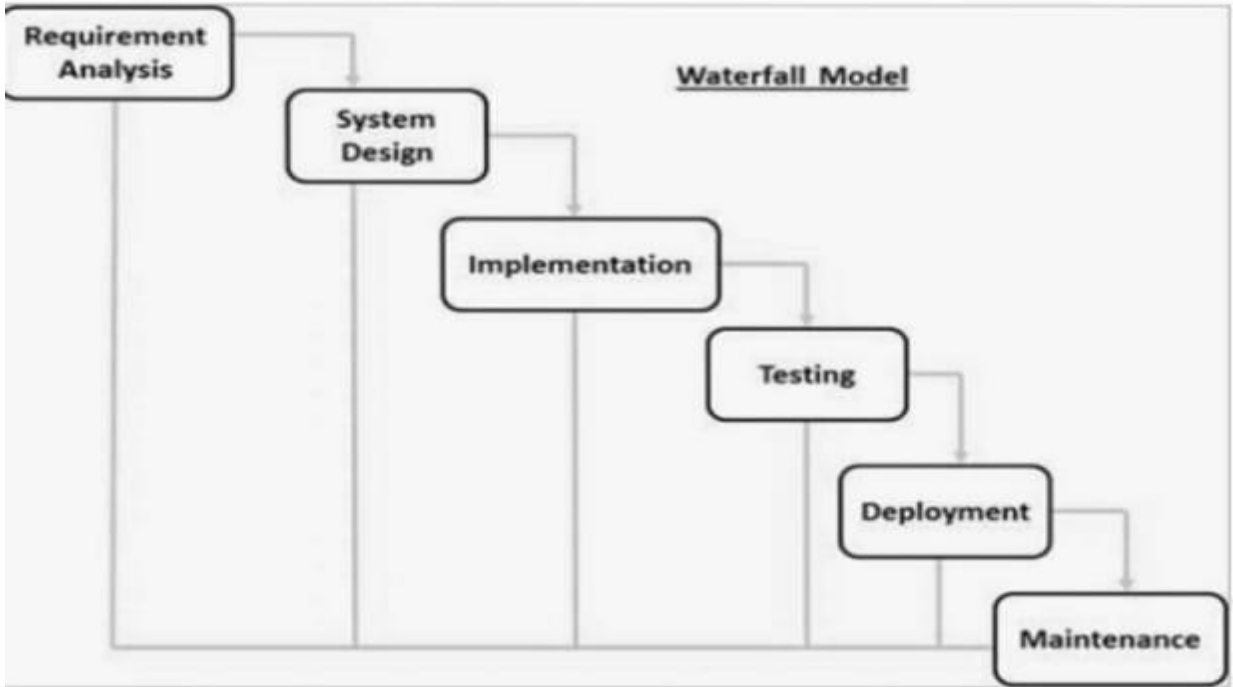
- The software engineering tool is an automated support for the software development.
- The tools are integrated i.e the information created by one tool can be used by the other tool.

For example: The Microsoft publisher can be used as a web designing tool

3(c)	Describe about Software MYTHS in detail.	5M	L2	C215.1
	<p><b>Ans:</b> Beliefs about software and the process used to build it- can be traced to the earliest days of computing myths have a number of attributes that have made them insidious.</p> <p><b>Management myths:</b> Managers with software responsibility, like managers in most disciplines, are often under pressure to maintain budgets, keep schedules from slipping, and improve quality.</p> <p><b>Myth:</b> We already have a book that's full of standards and procedures for building software - Won't that provide my people with everything they need to know?</p> <p><b>Reality:</b> The book of standards may very well exist but, is it used? Are software practitioners aware of its existence? Does it reflect modern software engineering practice?</p> <p><b>Myth:</b> If we get behind schedule, we can add more programmers and catch up.</p> <p><b>Reality:</b> Software development is not a mechanistic process like manufacturing. As new people are added, people who were working must spend time educating the new comers, thereby reducing the amount of time spend on productive development effort. People can be added but only in a planned and well co-ordinated manner.</p>			

	<p><b>Myth:</b> If I decide to outsource the software project to a third party, I can just relax and let that firm built it.</p> <p><b>Reality:</b> If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsources software projects.</p> <p><b>Customer myths:</b> The customer believes myths about software because software managers and practitioners do little to correct misinformation.</p> <p>Myths lead to false expectations and ultimately, dissatisfaction with the developer.</p> <p><b>Myth:</b> A general statement of objectives is sufficient to begin with writing programs - we can fill in the details later.</p> <p><b>Reality:</b> Although a comprehensive and stable statement of requirements is not always possible, an ambiguous statement of objectives is recipe for disaster.</p> <p><b>Myth:</b> Project requirements continually change, but change can be easily accommodated because software is flexible.</p> <p><b>Reality:</b> It is true that software requirements change, but the impact of change varies with the time at which it is introduced and change can cause upheaval that requires additional resources and major design modification.</p> <p><b>Developer's myths:</b> Myths that are still believed by software practitioners: during the early days of software, programming was viewed as an art from old ways and attitudes die hard.</p> <p><b>Myth:</b> Once we write the program and get it to work, our jobs are done.</p> <p><b>Reality:</b> Someone once said that the sooner you begin writing code, the longer it'll take you to get done. Industry data indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer for the first time.</p> <p><b>Myth:</b> The only deliverable work product for a successful project is the working program.</p> <p><b>Reality:</b> A working program is only one part of a software configuration that includes many elements. Documentation provides guidance for software support.</p> <p><b>Myth:</b> software engineering will make us create voluminous and unnecessary documentation and will invariably slows down.</p> <p><b>Reality:</b> software engineering is not about creating documents. It is about creating quality. Better quality leads to reduced rework. And reduced rework results in faster delivery times.</p>	5M	L2	C215.1
3(d)	<p>Describe the framework activities in software engineering briefly.</p> <p><b>Ans. process framework:</b></p> <ul style="list-style-type: none"> <li>• Establish the foundation for a complete software process.</li> <li>• Identifies a small number of framework activities.</li> <li>• Applies to all software projects, regardless of size/complexity.</li> <li>• Also, set of umbrella activities.</li> <li>• Application across entire software process.</li> <li>• Each framework activities has set of software engineering actions.</li> <li>• Each software engineering action has collection of related tasks called task sets or work tasks.</li> </ul> <p><b>Generic process of framework:</b></p> <p>It is applicable to that vast majority of software projects.</p> <p><b>1) Communication:</b></p> <p>This framework activity involves heavy communication and collaboration with the customer</p>			

	<p>and encompasses requirements gathering and other related activities.</p> <p><b>2) Planning:</b></p> <p>This activity establishes a plan for the software engineering work that follows. It required, the work products to be produced , and a work schedule.</p> <p><b>3) Modeling:</b></p> <p>This activity encompasses the creation of models that allow the developer and customer to better understand software requirements and the design that will achieve those requirements . the modeling activity is composed of two software engineering action analysis and design.</p> <ul style="list-style-type: none"> <li>• Analysis encompasses a set of work tasks.</li> <li>• Design encompasses work tasks that create a design model.</li> </ul> <p><b>4) Construction:</b></p> <p>This activity combines code generation and the testing that is required to uncover the errors in the code.</p> <p><b>5) Deployment:</b></p> <p>The software is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation .</p> <p>These five generic framework activities can be used during the development of small programs the creation of large web applications and for the engineering of large , complex computer based system.</p>			
<b>3(e)</b>	Explain in detail the capability Maturity Model Integration (CMMI)?	<b>5M</b>	L2	C215.1
	<p>Ans. <b>THE CAPABILITY MATURITY MODEL INTEGRATION (CMMI):</b></p> <p>The CMMI represents a process meta-model in two different ways: • As a continuous model • As a staged model. Each process area is formally assessed against specific goals and practices and is rated according to the following capability levels.</p> <p><b>Level 0: Incomplete:</b> The process area is either not performed or does not achieve all goals and objectives defined by CMMI for level 1 capability.</p> <p><b>Level 1: Performed:</b> All of the specific goals of the process area have been satisfied. Work tasks required to produce defined work products are being conducted.</p> <p><b>Level 2: Managed:</b> All level 1 criteria have been satisfied. In addition, all work associated with the process area conforms to an organizationally defined policy; all people doing the work have access to adequate resources to get the job done; stakeholders are actively involved in the process area as required; all work tasks and work products are “monitored, controlled, and reviewed;</p> <p><b>Level 3: Defined:</b> All level 2 criteria have been achieved. In addition, the process is “tailored from the organizations set of standard processes according to the organizations tailoring guidelines, and contributes and work products, measures and other process-improvement information to the organizational process assets”.</p> <p><b>Level 4: Quantitatively managed:</b> All level 3 criteria have been achieved. In addition, the process area is controlled and improved using measurement and quantitative assessment. ”Quantitative objectives for quality and process performance are established and used as criteria in managing the process”</p> <p><b>Level 5: Optimized:</b> All level 4 criteria have been achieved. In addition, the process area is adapted and optimized using quantitative means to meet changing customer needs and to continually improve the efficacy of the process area under consideration”</p>			

4(a)	Briefly explain Waterfall Model. Also write down advantages and disadvantages of the water fall model.	10M	L2	C215.1
	<p>Ans. <b>Waterfall Model :</b></p> <p>Waterfall approach was first SDLC Model to be used widely in Software Engineering. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially. The following illustration is a representation of the different phases of the Waterfall Model. The sequential phases in Waterfall model are –</p> <ul style="list-style-type: none"> <li>• <b>Requirement Gathering and analysis</b> – All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.</li> <li>• <b>System Design</b> – The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.</li> <li>• <b>Implementation</b> – With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.</li> <li>• <b>Integration and Testing</b> – All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.</li> <li>• <b>Deployment of system</b> – Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.</li> <li>• <b>Maintenance</b> – There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.</li> </ul> <p>All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model".</p>  <pre> graph TD     RA[Requirement Analysis] --&gt; SD[System Design]     SD --&gt; IM[Implementation]     IM --&gt; T[Testing]     T --&gt; D[Deployment]     D --&gt; M[Maintenance]   </pre> <p><b>Waterfall Model – Application</b></p>			

	<ul style="list-style-type: none"> <li>• Requirements are very well documented, clear and fixed.</li> <li>• Product definition is stable.</li> <li>• Technology is understood and is not dynamic.</li> <li>• There are no ambiguous requirements.</li> <li>• The project is short.</li> </ul> <p><b>Waterfall Model – Advantages</b></p> <ul style="list-style-type: none"> <li>• Simple and easy to understand and use</li> <li>• Easy to manage</li> <li>• Phases are processed and completed one at a time.</li> <li>• Works well for smaller projects</li> <li>• Clearly defined stages.</li> <li>• Well understood milestones.</li> </ul> <p><b>Waterfall Model – Disadvantages</b></p> <ul style="list-style-type: none"> <li>• No working software is produced until late during the life cycle.</li> <li>• High amounts of risk and uncertainty.</li> <li>• Not a good model for complex and object-oriented projects.</li> <li>• Poor model for long and ongoing projects.</li> </ul>			
4(b)	Explain Spiral Model with neat diagram, write advantages and disadvantages.	10M	L2	C215.1
	<p>It is the combination of waterfall model and iterative model. A scientist named Boehm proposed this Spiral model in the year 1986.</p> <p>The Spiral model is also named as metal model it is a risk driven model.</p> <p>The structure of Spiral model is represented as</p> <div data-bbox="305 1129 1200 1715" data-label="Diagram"> </div> <ul style="list-style-type: none"> <li>• The communication phase represent requirement analysis and planning represents risk analysis.</li> <li>• Modeling and construction phase represents the prototype building. Deployment phase represents the Evaluation of feedback/performance Evaluation.</li> </ul> <p>1. <b>Requirement Analysis :-</b> In this phase it analysis and expands the reason for what it has to bedone and additionally solutions are identified if incase there is a failure in the attempted version.</p> <p>2. <b>Risk Analysis :-</b></p>			



	<p>This phase focuses on the risk and it analysis the risks of all the possible solutions .This is performed to identify the faults are occurring in the process each risk is resolved using the most efficient strategy.</p> <p><b>3. Prototype Building :-</b> This phase is focuses o the building or developing the actual software and testing the software. In this phase the programmers create an architectural design, modules, physical product design and finalizes the design it takes proposal from the first two phases and turn it into usable software.</p> <p><b>4.Evaluation feedback/performance Evaluation :-</b> This is the final phase where the performance of the newly developed software gets tested and Evaluated programmers analyse their past work to learn before starting a new project. After this step they can start planning for the next phase and they can repeat the cycle in the end the software company releases the feedback into the market.</p> <p><b>Advantages of Spiral Model :-</b></p> <ul style="list-style-type: none"> <li>• Suitable for large scale product.</li> <li>• Development is fast and efficient cost estimator.</li> <li>• Proper risk management.</li> <li>• Involves customer feedback.</li> </ul> <p><b>Disadvantages of Spiral Model:-</b></p> <ul style="list-style-type: none"> <li>• Spiral way goon indefinitely and end of the project may not be known early.</li> <li>• more complex and expensive than other SDLC Models</li> </ul>	10M	L2	C215.1
4(c)	<p>Explain SDLC - Agile Model in brief and mention the advantages</p> <p><b>Agile Method for SDLC :-</b> Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customers and customers satisfaction by rapid delivery of working software product.</p> <p>Agile methods break the product into small incremental builds .These builds are provided in iterations. Each iteration typically lasts from about one to three weeks. Every iteration involves cross functional tzams working simultaneously on various areas like</p> <ul style="list-style-type: none"> <li>• Planning</li> <li>• Requirement analysis</li> <li>• Design</li> <li>• Coding</li> <li>• Unit testing and</li> <li>• Acceptance testing</li> </ul> <p>At the end of the iteration, a working product is display to the customer and the all important stake holders.</p>			



### Phases of Agile Model :-

Following are the phases in the Agile model are:-

- a. Requirements gathering
- b. Design the requirements
- c. Construction / iteration
- d. Testing /Quality Assurance
- e. Deploying
- f. Feedback

#### 1.Requirements Gathering:-

In this phase define the requirements you should business opportunities and plan the time and effort needed to build the project. Based on this information , you can evaluate technical and economic feasibility.

#### 2.Design the requirements:-

When you have identified the project work with stakeholders to define the requirements you can use the user flow diagram or the high level UML diagram to show the work of new features and show how it will apply to your existing system.

#### 3.Construction /iteration :-

When the team defines the requirements , the work begins. Designers and the developers start working on their projects , which aims to deploy a working product. The product will undergo various stages of improvement. So it includes simple, minimal functionality.

#### 4.Testing:-

In this phase ,the quality assurance team examines the products performance and looks for the bug.

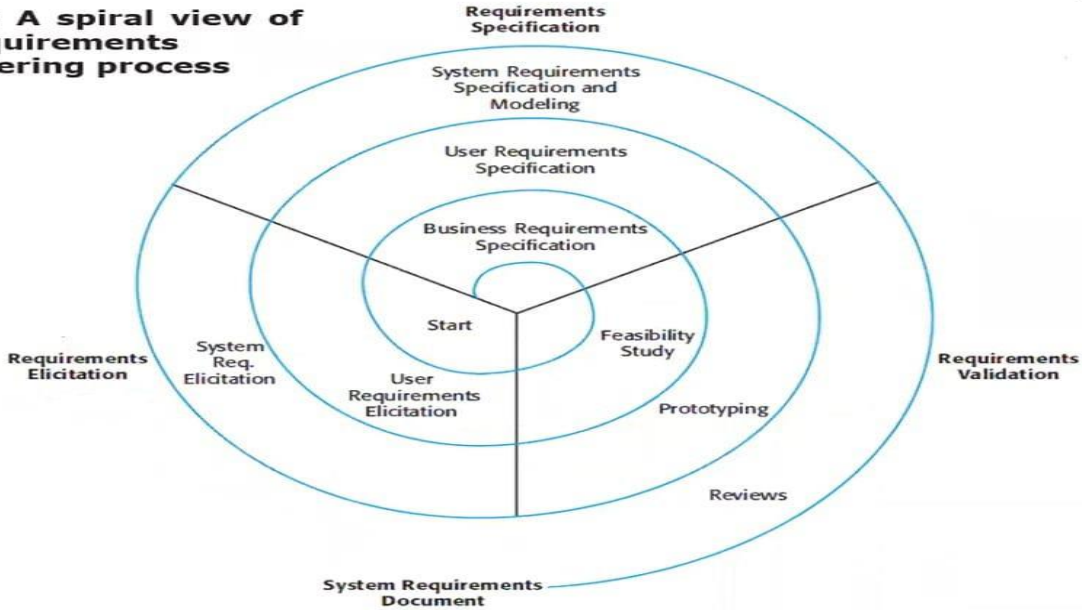
#### 5.Deployment:-

In this phase , the team issues a product for the users work environment .

#### 6.Feedback:-

After realizing the product , the last step is feedback. In this , the team receives feedback about the product and works through the feedback.

Q. No	Question (s)	Marks	BL	CO																
UNIT - II																				
1(a)	Define software requirements.	1M	L1	C215.2																
	Ans. The Features or Functions or Services that are expected by user in the software or product or system are called software requirements.																			
1(b)	List the types of software requirements.	1M	L1	C215.2																
	Ans. Depending upon the functionality or working the software requirements are classified into two types those are 1. Functional requirements 2. Non Functional requirements. Depending upon the peoples readability those are classified as two types. Those are, 1. User requirements 2. System requirements																			
1(c)	List the types of Non-functional requirements.	1M	L1	C215.2																
	Ans. The Non Functional requirements are classified into 3 types those are: 1. Product requirements 2. Organization requirements 3. External requirements.																			
1(d)	Define requirement engineering process.	1M	L1	C215.2																
	Ans. The process involved in gathering requirements, documenting requirements ,validating Requirements and managing requirements is called as requirement engineering process																			
1(e)	Mention the types of feasibility.	1M	L1	C215.2																
	Ans. The Feasibility study is classified into five types .those are 1. Economical feasibility 2. Operational feasibility 3. Legal feasibility 4. Technical feasibility 5. Schedule feasibility.																			
2(a)	Mention the main differences between the functional & non-functional requirements.	3M	L2	C215.2																
	<table><tr><th>Functional Testing</th><th>Non - Functional Testing</th></tr><tr><td>It is executed to analyze the functionality of components of an application as per the client's requirements.</td><td>It is executed to check the performance, reliability, scalability, and other non-functional aspects of an application.</td></tr><tr><td>It is executed in the early stages of development.</td><td>It is generally performed after functional testing.</td></tr><tr><td>Can be performed both manually and with automation tools.</td><td>Requires automation tools for effective testing.</td></tr><tr><td>Focuses on user requirements.</td><td>Focuses on user expectations.</td></tr><tr><td>Determines what the product is capable of</td><td>Determines how effectively the product works</td></tr><tr><td>Business requirements are the inputs of functional testing.</td><td>Parameters like speed, scalability are the inputs of non-functional testing.</td></tr><tr><td>Examples of Functional Testing: Unit Testing, White Box Testing, Smoke Testing, Sanity Testing, Usability Testing, Regression Testing.</td><td>Examples of Non-Functional Testing: Performance Testing, Load Testing, Stress Testing, Security Testing, Installation Testing, Cross Browser Compatibility Testing.</td></tr></table>				Functional Testing	Non - Functional Testing	It is executed to analyze the functionality of components of an application as per the client's requirements.	It is executed to check the performance, reliability, scalability, and other non-functional aspects of an application.	It is executed in the early stages of development.	It is generally performed after functional testing.	Can be performed both manually and with automation tools.	Requires automation tools for effective testing.	Focuses on user requirements.	Focuses on user expectations.	Determines what the product is capable of	Determines how effectively the product works	Business requirements are the inputs of functional testing.	Parameters like speed, scalability are the inputs of non-functional testing.	Examples of Functional Testing: Unit Testing, White Box Testing, Smoke Testing, Sanity Testing, Usability Testing, Regression Testing.	Examples of Non-Functional Testing: Performance Testing, Load Testing, Stress Testing, Security Testing, Installation Testing, Cross Browser Compatibility Testing.
Functional Testing	Non - Functional Testing																			
It is executed to analyze the functionality of components of an application as per the client's requirements.	It is executed to check the performance, reliability, scalability, and other non-functional aspects of an application.																			
It is executed in the early stages of development.	It is generally performed after functional testing.																			
Can be performed both manually and with automation tools.	Requires automation tools for effective testing.																			
Focuses on user requirements.	Focuses on user expectations.																			
Determines what the product is capable of	Determines how effectively the product works																			
Business requirements are the inputs of functional testing.	Parameters like speed, scalability are the inputs of non-functional testing.																			
Examples of Functional Testing: Unit Testing, White Box Testing, Smoke Testing, Sanity Testing, Usability Testing, Regression Testing.	Examples of Non-Functional Testing: Performance Testing, Load Testing, Stress Testing, Security Testing, Installation Testing, Cross Browser Compatibility Testing.																			

2(b)	List the readers of user requirements, system requirements, software requirement specification document.	3M	L1	C215.2
	<p><b>Ans.</b>  <u>Readers of user requirements:</u>  The readers of user requirements are,  -&gt;System manager  -&gt; Client managers  -&gt; System architects  -&gt; Client engineers  <u>Readers of system requirements:</u>  The readers of system requirements are,  -&gt; Client manager  -&gt;Contract manager  -&gt; System architects  -&gt;Software developers.  <u>Readers of software requirement specification document:</u>  The readers of software requirement specification document are,  -&gt; System customers  -&gt; Managers  -&gt; System Engineers  -&gt; System Test Engineers  -&gt; System Maintenance Engineers</p>			
2(c)	Draw the structure of spiral view of requirement engineering process.	3M	L1	C215.2
	<p><b>Ans.</b> In the spiral view of requirement engineering process we are having three phases those are:</p> <ol style="list-style-type: none"> <li>1. Requirement elicitation</li> <li>2. Requirement specification</li> <li>3. Requirement validation</li> </ol> <p>The structure of spiral model of requirement engineering process is represented below</p> <p><b>Figure: A spiral view of the requirements engineering process</b></p> 			

2(d)	What are the goals of feasibility study?	3M	L1	C215.2																								
	<b>Ans.</b> Goals of Feasibility Study: 1. Feasibility study is the process of accessing the strengths and weakness of the software or project or system. 2. It prevents the directions of activities that will improve a projector software or system and achieve the desired output. 3. It determines whether the projects is technically financially, legally and operationally feasible with in the estimated cost and time.																											
2(e)	Mention the process activities involved in requirement elicitation & analysis.	3M	L1	C215.2																								
	<b>Ans.</b> There are four process activities involved in requirement elicitation & analysis. Those are, 1. Requirement Discovery 2. Requirement Classification & Organization 3. Requirement Prioritization & Negotiation 4. Requirement Specification																											
3(a)	Compare and contrast functional and non-functional requirements in software engineering.	5M	L2	C215.2																								
	<table><tr><td>S. No</td><td>Functional Requirements</td><td>Non – Functional Requirements</td></tr><tr><td>1.</td><td>The Functional Requirements concentrates on what to do must in a software or product or system.</td><td>The Non – Functional Requirements concentrates on quality attributes of the product or system or software.</td></tr><tr><td>2.</td><td>The Functional Requirements helps in identify the functions or features or services provided by the software or product or system.</td><td>The Non – Functional Requirements helps in identify performance, security, privacy, usability and scalability parameters.</td></tr><tr><td>3.</td><td>Easy to maintain the functional requirements of a software or product or system.</td><td>Not easy to maintain i.e. difficult to maintain the non – functional requirements of a software or product or system.</td></tr><tr><td>4.</td><td>These are the mandatory requirements.</td><td>These are the non – mandatory requirements.</td></tr><tr><td>5.</td><td>These are designed depending on the users requirements.</td><td>These are designed depending on the expectation of the users.</td></tr><tr><td>6.</td><td>These are the requirements specified by the users.</td><td>These are the requirements specified by the software developers, software architects, technical persons.</td></tr><tr><td>7.</td><td>It describes what the product does.</td><td>It describes how the product does.</td></tr></table>				S. No	Functional Requirements	Non – Functional Requirements	1.	The Functional Requirements concentrates on what to do must in a software or product or system.	The Non – Functional Requirements concentrates on quality attributes of the product or system or software.	2.	The Functional Requirements helps in identify the functions or features or services provided by the software or product or system.	The Non – Functional Requirements helps in identify performance, security, privacy, usability and scalability parameters.	3.	Easy to maintain the functional requirements of a software or product or system.	Not easy to maintain i.e. difficult to maintain the non – functional requirements of a software or product or system.	4.	These are the mandatory requirements.	These are the non – mandatory requirements.	5.	These are designed depending on the users requirements.	These are designed depending on the expectation of the users.	6.	These are the requirements specified by the users.	These are the requirements specified by the software developers, software architects, technical persons.	7.	It describes what the product does.	It describes how the product does.
S. No	Functional Requirements	Non – Functional Requirements																										
1.	The Functional Requirements concentrates on what to do must in a software or product or system.	The Non – Functional Requirements concentrates on quality attributes of the product or system or software.																										
2.	The Functional Requirements helps in identify the functions or features or services provided by the software or product or system.	The Non – Functional Requirements helps in identify performance, security, privacy, usability and scalability parameters.																										
3.	Easy to maintain the functional requirements of a software or product or system.	Not easy to maintain i.e. difficult to maintain the non – functional requirements of a software or product or system.																										
4.	These are the mandatory requirements.	These are the non – mandatory requirements.																										
5.	These are designed depending on the users requirements.	These are designed depending on the expectation of the users.																										
6.	These are the requirements specified by the users.	These are the requirements specified by the software developers, software architects, technical persons.																										
7.	It describes what the product does.	It describes how the product does.																										

	8.	After completion of these functional requirements only the system allows to frame the non – functional requirements.	Without completion of functional requirements the system doesn't allow the non – functional requirements.	
<b>3(b)</b>	Explain regarding the user requirements & system requirements.		<b>5M</b>	L2 C215.2
	<p><b>Ans.</b> Depending on the persons readability the software requirements are classified into two types. Those are,</p> <ol style="list-style-type: none"> <li>1. User requirements</li> <li>2. System requirements</li> </ol> <p><b>1 User requirements:</b> User requirements are the statements in a natural language plus, diagrams of what services the system is expected to provide the users.</p> <p><b>Readers of the user requirements:</b></p> <ul style="list-style-type: none"> <li>-&gt;System manager</li> <li>-&gt; Client managers</li> <li>-&gt; System architects</li> <li>-&gt; Client engineers</li> </ul> <p><b>2 System requirements:</b> These are more detailed specifications of the software system functions or features or services. The system requirements document should be defined exactly what is to be implemented.</p> <p><b>Readers of the system requirements:</b></p> <ul style="list-style-type: none"> <li>-&gt; Client manager</li> <li>-&gt;Contract manager</li> <li>-&gt; System architects</li> <li>-&gt;Software developers</li> </ul>			
<b>3(c)</b>	Explain the interface specifications briefly.		<b>5M</b>	L2 C215.2
	<p><b>Ans . -INTERFACE SPECIFICATION:</b> Most systems must operate with other systems and the operating interfaces must be specified as part of the requirements. Three types of interface may have to be defined.</p> <ol style="list-style-type: none"> <li>1. Procedural interfaces: Where existing programs or sub-systems offer a range of services that are accessed by calling interface procedures. These interfaces are sometimes called Application Programming Interfaces (APIs)</li> <li>2. Data Structures that are exchanged: that are passed from one sub-system to another. Graphical data models are the best notations for this type of description.</li> <li>3. Data representations: that have been established for an existing sub-system.</li> </ol>			
<b>3(d)</b>	Explain regarding the feasibility study in detail.		<b>5M</b>	L2 C215.2
	<p><b>Ans.</b> <b><u>Feasibility study:</u></b></p> <p>Feasibility study in requirement engineering process of Software engineering is a study that represents the evaluation of feasibility of the proposed software (or) project (or) system.</p> <p><b>Goals of Feasibility study:</b></p> <p>→Feasibility study is the process of accessing the strengths &amp; Weakness of the software (or)</p>			

project(or) System.

→It presents the directions of activities that will improve a project (3) software (or) System & achieve the desired output.

→ It determines whether the project is technically, financially, legally & operationally feasible within the estimated cost & time

### **Types of feasibility:**

The Feasibility study is classified into five types. Those are,

1. Economical Feasibility
2. Operational Feasibility
3. Legal Feasibility
4. Technical Feasibility
5. Schedule Feasibility

#### **1) Economical Feasibility:**

→The Feasibility which deals with whether the system (or) project (or) Software Can be developed within the budget (or) not

#### **2) operational Feasibility:**

→This operational Feasibility is used to determine that how will the proposed system will be developed to solve the current problems.

#### **3) Legal Feasibility:**

→This legal Feasibility deals with the legal issues related to the system (or) project (or) Software such as cyber laws & copy rights.

#### **4) Technical Feasibility:**

→The Technical Feasibility deals with the level & type of technology needed for the system (or) Software (or) project.

#### **5) Schedule Feasibility:**

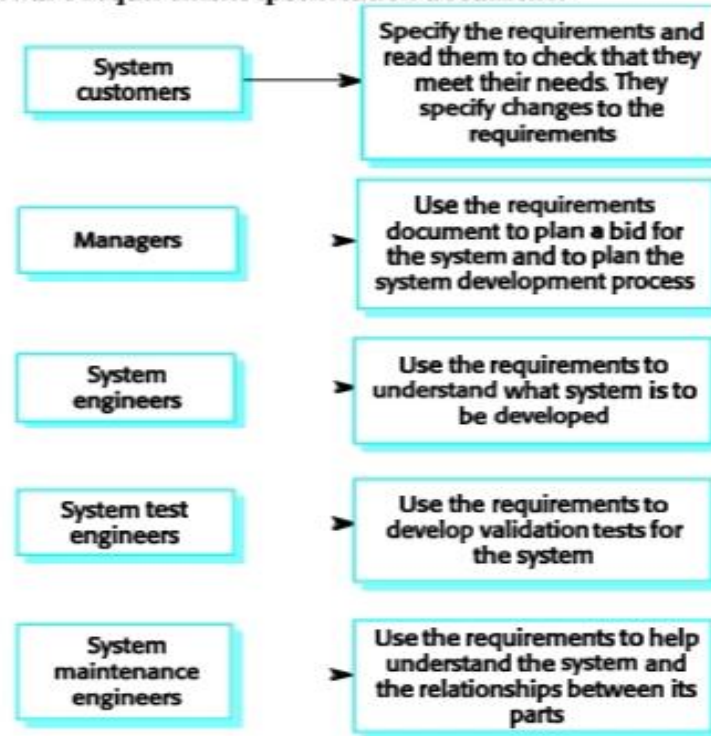
→The Schedule Feasibility deals with whether the System (or) project (or) Software can be developed within the given time period (or) not.

3(e)	Explain regarding the viewpoints in detail.	5M	L2	C215.2
	<p>Ans.</p> <p><b>Viewpoints:</b></p> <p>Viewpoints are a way of structuring the requirements to represent the perspectives of different stakeholders. There are three types of viewpoints in the requirement elicitation techniques.</p> <p>Those are,</p> <ol style="list-style-type: none"> <li>1. Direct Viewpoints or Interactor Viewpoints</li> <li>2. Indirect Viewpoints</li> <li>3. Domain Viewpoints</li> </ol> <p><b>1. Direct Viewpoints:</b> Direct viewpoints involve stakeholders who will directly interact with the system. These individuals provide firsthand requirements based on their experiences and needs.</p> <p><b>2. Indirect Viewpoints:</b> Indirect viewpoints represent stakeholders who do not use the system but influence the requirements. Their insights are crucial for understanding Broader needs and constraints.</p> <p><b>3. Domain Viewpoints:</b> Domain viewpoints reflect the characteristics and constraints of the Environment in which the system will operate. These include considerations from existing systems or manual processes being replaced.</p>			
4(a)	Explain the purpose and key components of a Software Requirements Specification (SRS) document. Identify the different readers of an SRS and their concerns. And also describe the relevant IEEE standards for SRS documents.	10M	L2	C215.2
	<p>Ans. THE SOFTWARE REQUIREMENT SPECIFICATION DOCUMENT (SRS):</p> <ul style="list-style-type: none"> <li><input type="checkbox"/> A software requirement specifications (SRS) document is a detailed description of what a software will do and how it should perform.</li> <li><input type="checkbox"/> A software requirement specifications (SRS) document lists the requirements, expectations, design, and standards for a future project. These include the high-level</li> </ul>			



business requirements dictating the goal of the project, end-user requirements and needs, and the product's functionality in technical terms. To put it simply, an SRS provides a detailed description of how a software product should work and how your development team should make it work.

**Readers of software requirement specification document:**



IEEE requirements standard defines a generic structure for a requirement specification document that must be instantiated for each specific system.

1. Introduction.

- i) Purpose of the requirements document
- ii) Scope of the project
- iii) Definitions, acronyms and abbreviations
- iv) References
- v) Overview of the remainder of the document

2. General description.

- i) Product perspective

	ii) Product functions iii) User characteristics iv) General constraints v) Assumptions and dependencies 3. Specific requirements cover functional, non-functional and interface requirements. The requirements may document external interfaces, describe system functionality and performance, specify logical database requirements, design constraints, emergent system properties and quality characteristics. 4. Appendices. 5. Index			
<b>4(b)</b>	Explain about the requirements elicitation techniques in detail.	<b>10M</b>	L2	C215.2
<b>Ans</b>	<p>Requirements Elicitation techniques:</p> <p>The requirements elicitation techniques are,</p> <ul style="list-style-type: none"> <li>➤ Viewpoints</li> <li>➤ Interviews</li> <li>➤ Scenarios</li> <li>➤ Use Cases</li> <li>➤ Surveys and Questionnaires</li> <li>➤ Brainstorming and Prototyping</li> </ul> <p><b><u>Viewpoints:</u></b></p> <p>Viewpoints are a way of structuring the requirements to represent the perspectives of different stakeholders. There are three types of viewpoints in the requirement elicitation techniques.</p> <p>Those are,</p> <p>1. Direct Viewpoints or Interactor Viewpoints</p>			

## 2. Indirect Viewpoints

## 3. Domain Viewpoints

**1. Direct Viewpoints:** Direct viewpoints involve stakeholders who will directly interact with the system. These individuals provide firsthand requirements based on their experiences and needs.

**2. Indirect Viewpoints:** Indirect viewpoints represent stakeholders who do not use the system but influence the requirements. Their insights are crucial for understanding broader needs and constraints.

**3. Domain Viewpoints:** Domain viewpoints reflect the characteristics and constraints of the environment in which the system will operate. These include considerations from existing systems or manual processes being replaced.

### Interviewing:

In formal or informal interviewing, the Requirements Elicitation team puts questions to stakeholders about the system that they use and the system to be developed.

There are two types of interviews.

**Closed interviews** where a pre-defined set of questions are answered.

**Open interviews** where there is no pre-defined agenda and a range of issues are explored with stakeholders.

### Scenarios:

Scenarios are real-life examples of how a system can be used.

They should include

A description of the starting situation;

A description of the normal flow of events;

A description of what can go wrong;

Information about other concurrent activities;

A description of the state when the scenario finishes.

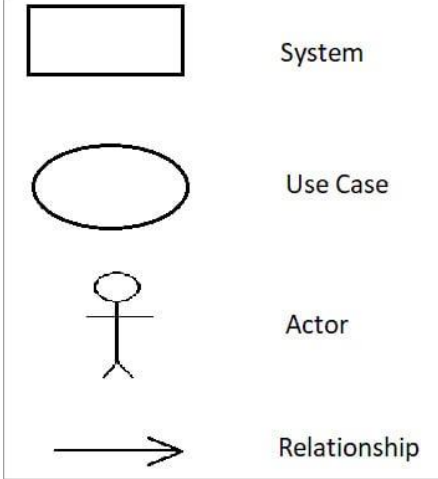
### Use cases:

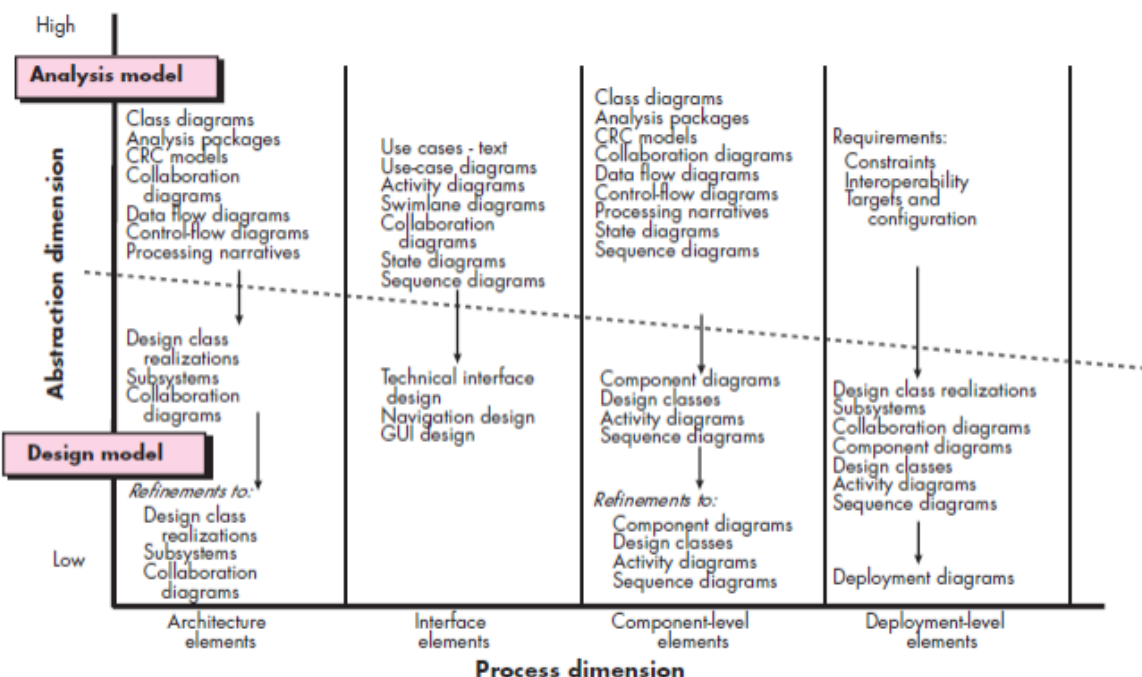
Use-cases are a scenario based technique in the UML which identify the actors in an interaction and which describe the interaction itself.

	<p>A set of use cases should describe all possible interactions with the system.</p> <p>Sequence diagrams may be used to add detail to use-cases by showing the sequence of event processing in the system.</p>			
4(c)	Define requirements validation in the requirements engineering process. Explain its importance and briefly describe techniques used to validate requirements.	10M	L2	C215.2
<p><b>Ans.</b></p> <p><b><u>Requirements Validation:</u></b></p> <ul style="list-style-type: none"> <li>Concerned with demonstrating that the requirements define the system that the customer really wants.</li> <li>Requirements error costs are high so validation is very important             <ul style="list-style-type: none"> <li>- Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.</li> </ul> </li> </ul> <p><b><u>Requirements checking:</u></b></p> <p><b>Validity:</b> Does the system provide the functions which best support the customer's needs?</p> <p><b>Consistency:</b> Are there any requirements conflicts?</p> <p><b>Completeness:</b> Are all functions required by the customer included?</p> <p><b>Realism:</b> Can the requirements be implemented given available budget and technology</p> <p><b>Verifiability:</b> Can the requirements be checked?</p> <p><b><u>Requirements validation techniques:</u></b></p> <p><b>Requirements reviews:</b></p> <p>1) Systematic manual analysis of the requirements.</p> <p><b>Prototyping:</b></p> <p>1) Using an executable model of the system to check requirements.</p> <p><b>Test-case generation:</b></p> <p>1) Developing tests for requirements to check testability.</p> <p><b><u>Requirements reviews:</u></b></p> <p>1) Regular reviews should be held while the requirements definition is being formulated.</p>				

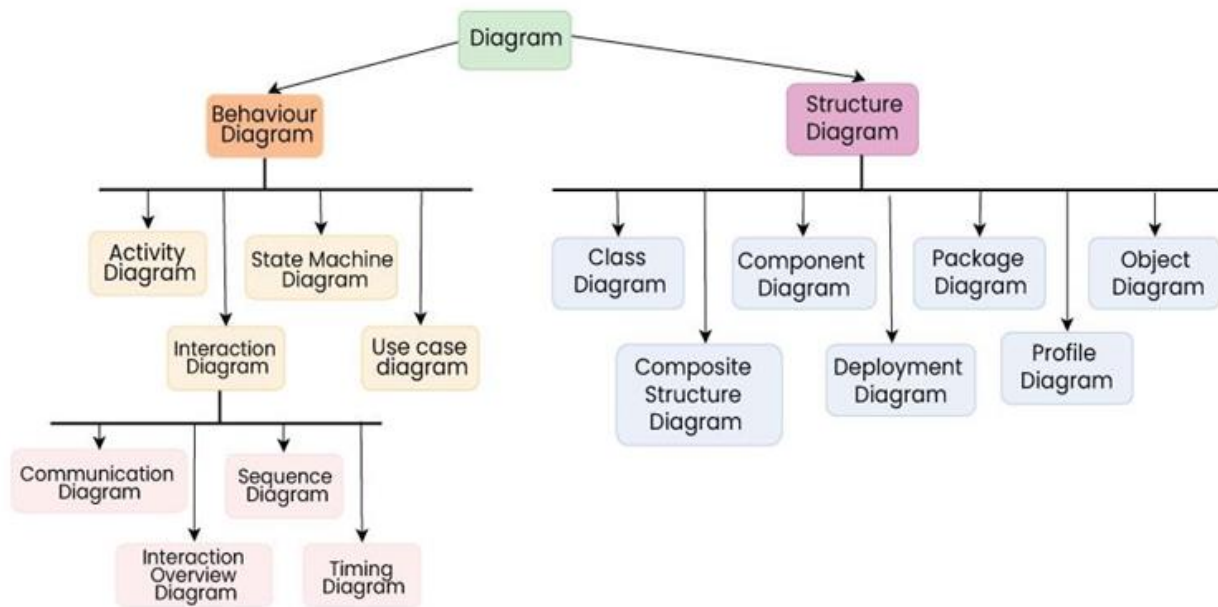
	<p>2)Both client and contractor staff should be involved in reviews.</p> <p>3) Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage.</p>
--	--

Q. No	Question (s)	Marks	BL	CO
<b>UNIT – III</b>				
<b>1(a)</b>	Define design engineering.	<b>1M</b>	L1	C215.3
	Ans. It encompasses the set of principles, concepts, and practices that lead to the development of a high- quality system or product.			
<b>1(b)</b>	What is the goal of design engineering?	<b>1M</b>	L1	C215.3
	Ans. The goal of design engineering is to produce a model or representation that exhibits firmness, commodity.			
<b>1(c)</b>	List out Quality attributes.	<b>1M</b>	L1	C215.3
	Ans. The quality attributes are, 1. Functionality 2. Usability 3. Reliability 4. Performance 5. Supportability			
<b>1(d)</b>	Define Software Architecture.	<b>1M</b>	L1	C215.3
	Software architecture is the structure or organization of program components (modules), the manner in which these components interact, and the structure of data that are used by the components.			
<b>1(e)</b>	What is data design?	<b>1M</b>	L1	C215.3
	The data design action translates the data objects defined as part of analysis model into data structures at the software component level or a data architecture at the application levels.			
<b>2(a)</b>	List out any four design concepts.	<b>3M</b>	L1	C215.3
	Ans.(i) Abstraction (ii) Architecture (iii) Patterns (iv) Modularity			
<b>2(b)</b>	Define the concept of “use case”.	<b>3M</b>	L1	C215.3
	Ans. A use case describes the interactions between one or more Actors and the system in order to provide an observable result.			
<b>2(c)</b>	Give the conceptual model of UML.	<b>3M</b>	L2	C215.3
	<p><b>THE THREE ASPECTS OF UML</b></p> <ul style="list-style-type: none"> <li><b>MODEL</b>: it is a representation of a subject it captures a set of ideas (known as abstractions) about its subject</li> <li><b>LANGUAGE</b>: enables us to communicate about a subject which includes the requirements and the system it is difficult to communicate and collaborate for a team to successfully develop a system without a language</li> <li><b>UNIFIED</b>: to bring together the information systems and technology industry's best engineering practices these practices involve applying techniques that allow us to successfully develop systems</li> </ul>			

2(d)	What are component diagrams?	3M	L2	C215.3
	A Component-Based Diagram, often called a Component Diagram, is a type of structural diagram in the Unified Modeling Language (UML) that visualizes the organization and interrelationships of the components within a system.			
2(e)	What are the use-case diagram notations?	3M	L2	C215.3
	 <p>The diagram illustrates the standard notations for use-case diagrams in UML. It consists of four items arranged vertically, each with a symbol and a label to its right:</p> <ul style="list-style-type: none"> <li>A rectangle labeled "System".</li> <li>An oval labeled "Use Case".</li> <li>A stick figure labeled "Actor".</li> <li>A horizontal arrow labeled "Relationship".</li> </ul>			
3(a)	Briefly Explain Quality attributes.	5M	L2	C215.3
	<p>Ans.</p> <p>The FURPS quality attributes represent a target for all software design:</p> <ul style="list-style-type: none"> <li>➤ <b>Functionality</b> is assessed by evaluating the feature set and capabilities of the program, the generality of the functions that are delivered, and the security of the overall system.</li> <li>➤ <b>Usability</b> is assessed by considering human factors, overall aesthetics, consistency and documentation.</li> <li>➤ <b>Reliability</b> is evaluated by measuring the frequency and severity of failure, the accuracy of output results, and the mean – time –to- failure (MTTF), the ability to recover from failure, and the predictability of the program.</li> <li>➤ <b>Performance</b> is measured by processing speed, response time, resource consumption, throughput, and efficiency</li> <li>➤ <b>Supportability</b> combines the ability to extend the program (extensibility), adaptability, serviceability- these three attributes represent a more common term maintainability</li> </ul> <p>Not every software quality attribute is weighted equally as the software design is developed. One application may stress functionality with a special emphasis on security.</p> <p>Another may demand performance with particular emphasis on processing speed</p> <p>A third might focus on reliability</p>			

3(b)	Explain regarding the Design Model.	5M	L2	C215.3
	<p>Ans. <b>THE DESIGN MODEL:</b></p> <p>The design model can be viewed in two different dimensions as illustrated in Figure 8.4. The process dimension indicates the evolution of the design model as design tasks are executed as part of the software process. The abstraction dimension represents the level of detail as each element of the analysis model is transformed into a design equivalent and then refined iteratively. Referring to below Figure, the dashed line indicates the boundary between the analysis and design models. The analysis model slowly blends into the design and a clear distinction is less obvious. The elements of the design model use UML diagrams, that were used in the analysis model. The difference is that these diagrams are refined and elaborated as part of design; more implementation-specific detail is provided, and architectural structure and style, components that reside within the architecture, and interfaces between the components and with the outside world are all emphasized.</p> <p>You should note, however, that model elements indicated along the horizontal axis are not always developed in a sequential fashion. The deployment model is usually delayed until the design has been fully developed.</p> 			
3(c)	Give the UML diagrams Classification chart.	5M	L2	C215.3





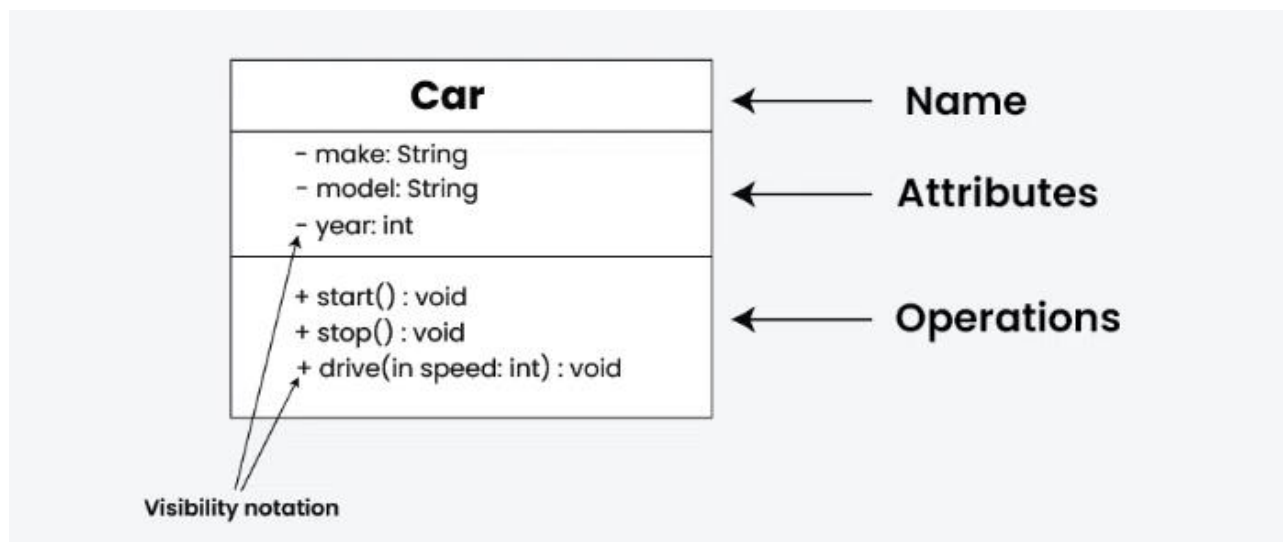
**3(d)** Explain the UML class notations.

**5M**

L2

C215.3

Class notation is a graphical representation used to depict classes and their relationships in object-oriented modeling.



**1. Class Name:**


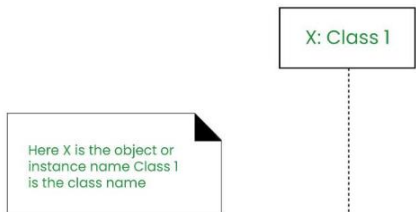
- The name of the class is typically written in the top compartment of the class box and is centered and bold.

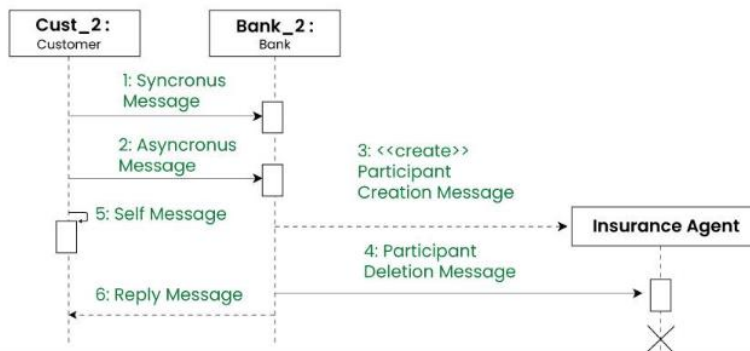
**2. Attributes:**

- Attributes, also known as properties or fields, represent the data members of the class. They are listed in the second compartment of the class box and often include the visibility (e.g., public, private) and the data type of each attribute.

**3. Methods:**

- Methods, also known as functions or operations, represent the behavior or functionality of the class. They are listed in the third compartment of the class box and include the visibility (e.g., public, private), return type, and parameters of each method.

	<p><b>4. Visibility Notation:</b></p> <ul style="list-style-type: none"> <li>• Visibility notations indicate the access level of attributes and methods. Common visibility notations include: <ul style="list-style-type: none"> <li>○ + for public (visible to all classes)</li> <li>○ - for private (visible only within the class)</li> <li>○ # for protected (visible to subclasses)</li> <li>○ ~ for package or default visibility (visible to classes in the same package)</li> </ul> </li> </ul>			
<b>3(e)</b>	Explain the sequence diagram notations.	<b>5M</b>	L2	C215.3
	<p><b><u>Sequence Diagram Notations</u></b></p> <p><b>1. Actors</b></p> <p>An actor in a UML diagram represents a type of role where it interacts with the system and its objects. It is important to note here that an actor is always outside the scope of the system we aim to model using the UML diagram.</p>  <p>We use actors to depict various roles including human users and other external subjects. We represent an actor in a UML diagram using a stick person notation. We can have multiple actors in a sequence diagram.</p> <p><b>2. Lifelines</b></p> <p>A lifeline is a named element which depicts an individual participant in a sequence diagram. So basically each instance in a sequence diagram is represented by a lifeline. Lifeline elements are located at the top in a sequence diagram. The standard in UML for naming a lifeline follows the following format:</p> <p><i>Instance Name : Class Name</i></p>  <p><b>3. Messages</b></p> <p>Communication between objects is depicted using messages. The messages appear in a sequential order on the lifeline.</p> <ul style="list-style-type: none"> <li>• We represent messages using arrows.</li> <li>• Lifelines and messages form the core of a sequence diagram.</li> </ul>			

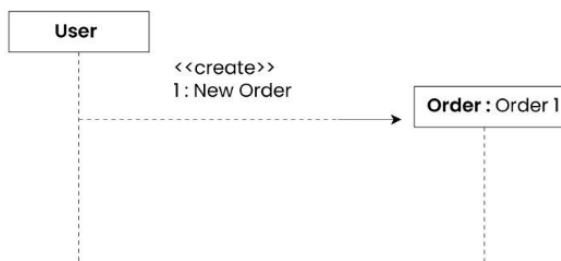


Messages can be broadly classified into the following categories:

1. Synchronous messages
2. Asynchronous messages

#### 4. Create message

We use a Create message to instantiate a new object in the sequence diagram. There are situations when a particular message call requires the creation of an object. It is represented with a dotted arrow and create word labelled on it to specify that it is the create Message symbol.



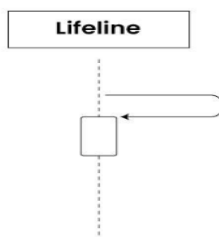
#### 5. Delete Message

We use a Delete Message to delete an object. When an object is deallocated memory or is destroyed within the system we use the Delete Message symbol. It destroys the occurrence of the object in the system. It is represented by an arrow terminating with a x.



#### 6. Self Message

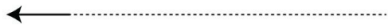
Certain scenarios might arise where the object needs to send a message to itself. Such messages are called Self Messages and are represented with a **U shaped arrow**.



#### 7. Reply Message

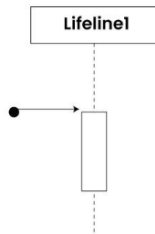
Reply messages are used to show the message being sent from the receiver to the sender. We

represent a return/reply message using an **open arrow head with a dotted line**. The interaction moves forward only when a reply message is sent by the receiver.



### 8. Found Message

A Found message is used to represent a scenario where an unknown source sends the message. It is represented using an **arrow directed towards a lifeline** from an end point.



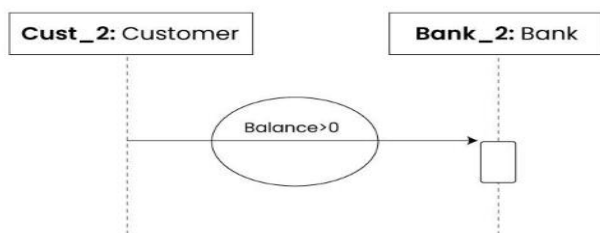
### 9. Lost Message

A Lost message is used to represent a scenario where the recipient is not known to the system. It is represented using an arrow directed towards an end point from a lifeline.



### 10. Guards

To model conditions we use guards in UML. They are used when we need to restrict the flow of messages on the pretext of a condition being met. Guards play an important role in letting software developers know the constraints attached to a system or a particular process.



4(a)	Explain regarding the following design concepts briefly. (i) Abstraction, (ii) Architecture, (iii) Patterns, (iv) Modularity	10M	L2	C215.3
	<p>Ans. <b>Design concepts</b> has evolved over the history of software engineering. Each concept provides the software designer with a foundation from which more sophisticated design methods can be applied. Some of the software design concepts that span both traditional and object-oriented software development is given below.</p> <p><b>(i) Abstraction:</b> When you consider a modular solution to any problem, many levels of abstraction can be posed. At the highest level of abstraction, a solution is stated in broad terms using the language of the problem environment. At lower levels of abstraction, a more detailed description of the solution is provided. Finally, at the lowest level of abstraction, the solution is stated in a manner that can be directly implemented. A <b>procedural abstraction</b> refers to a sequence of</p>			

instructions that have a specific and limited function. The name of a procedural abstraction implies these functions, but specific details are suppressed. A **data abstraction** is a named collection of data that describes a data object.

**(ii) Architecture:** Software architecture alludes to “the overall structure of the software and the ways in which that structure provides conceptual integrity for a system”. In its simplest form, architecture is the structure or organization of program components (modules), the manner in which these components interact, and the structure of data that are used by the components. One goal of software design is to derive an architectural rendering of a system. A set of architectural patterns enables a software engineer to solve common design problems.

Shaw and Garlan describe a set of properties as part of an architectural design:

**Structural properties:** This aspect of the architectural design representation defines the components of a system (e.g., modules, objects, filters) and the manner in which those components are packaged and interact with one another. For example, objects are packaged to encapsulate both data and the processing that manipulates the data and interact via the invocation of methods.

**Extra-functional properties:** The architectural design description should address how the design architecture achieves requirements for performance, capacity, reliability, security, adaptability, and other system characteristics.

**Families of related systems:** The architectural design should draw upon repeatable patterns that are commonly encountered in the design of families of similar systems. In essence, the design should have the ability to reuse architectural building blocks.

**(iii) Patterns:** A pattern is a named nugget of insight which conveys the essence of a proven solution to a recurring problem within a certain context amidst competing concerns. Stated A design pattern describes a design structure that solves a particular design problem within a specific context and amid “forces” that may have an impact on the manner in which the pattern is applied and used.

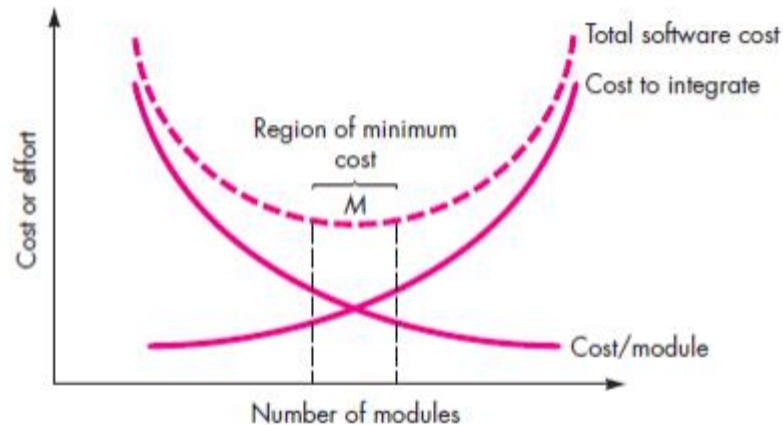
The intent of each design pattern is to provide a description that enables a designer to determine

(1) whether the pattern is applicable to the current work

(2) whether the pattern can be reused (hence, saving design time)

(3) whether the pattern can serve as a guide for developing a similar, but functionally or structurally different pattern.

**(iv) Modularity:** Modularity is the most common manifestation of separation of concerns. Software is divided into separately named and addressable components, sometimes called modules, that are integrated to satisfy problem requirements. It has been stated that “modularity is the single attribute of software that allows a program to be intellectually manageable”. The number of control paths, span of reference, number of variables, and overall complexity would make understanding close to impossible. In almost all instances, you should break the design into many modules, hoping to make understanding easier and, as a consequence, reduce the cost required to build the software. if you subdivide software indefinitely the effort required to develop it will become negligibly small! Unfortunately, other forces come into play, causing this conclusion to be (sadly) invalid. Referring to below figure, the effort (cost) to develop an individual software module does decrease as the total number of modules increases.



Given the same set of requirements, more modules means smaller individual size. However, as the number of modules grows, the effort (cost) associated with integrating the modules also grows. These characteristics lead to a total cost or effort curve shown in the figure. There is a number,  $M$ , of modules that would result in minimum development cost, but we do not have the necessary sophistication to predict  $M$  with assurance.

The curves shown in above figure do provide useful qualitative guidance when modularity is considered. You should modularize, but care should be taken to stay in the vicinity of  $M$ . **Undermodularity** or **overmodularity** should be avoided.

You modularize a design (and the resulting program) so that development can be more easily planned; software increments can be defined and delivered; changes can be more easily accommodated; testing and debugging can be conducted more efficiently, and long-term maintenance can be conducted without serious side effects.

4(b)	Discuss data design in detail.	10M	L2	C215.3
	<p>The <i>data design</i> action translates data objects defined as part of the analysis model (Chapter 8) into data structures at the software component level and, when necessary, a database architecture at the application level. In some situations, a database must be designed and built specifically for a new system. In others, however, one or more existing databases are used.</p> <p><b>10.2.1 Data Design at the Architectural Level</b></p> <p>Today, businesses large and small are awash in data. It is not unusual for even a moderately sized business to have dozens of databases serving many applications encompassing hundreds of gigabytes of data. The challenge is to extract useful information from this data environment, particularly when the information desired is cross-functional (e.g., information that can be obtained only if specific marketing data are cross-correlated with product engineering data).</p>			

To solve this challenge, the business IT community has developed *data mining* techniques, also called *knowledge discovery in databases* (KDD), that navigate through existing databases in an attempt to extract appropriate business-level information. However, the existence of multiple databases, their different structures, the degree of detail contained with the databases, and many other factors make data mining difficult within an existing database environment. An alternative solution, called a *data warehouse*, adds an additional layer to the data architecture.

A data warehouse is a separate data environment that is not directly integrated with day-to-day applications but encompasses all data used by a business [MAT96]. In a sense, a data warehouse is a large, independent database that has access to the data that are stored in databases that serve the set of applications required by a business.

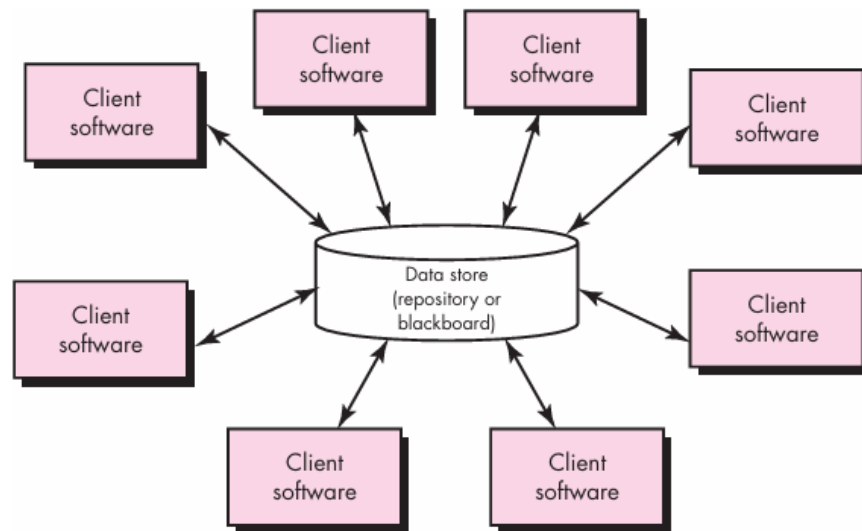
### 10.2.2 Data Design at the Component Level

Data design at the component level focuses on the representation of data structures that are directly accessed by one or more software components. Wasserman [WAS80] has proposed a set of principles that may be used to specify and design such data structures. In actuality, the design of data begins during the creation of the analysis model. Recalling that requirements analysis and design often overlap, we consider the following set of principles (adapted from [WAS80]) for data specification:

1. *The systematic analysis principles applied to function and behavior should also be applied to data.* Representations of data flow and content should also be developed and reviewed, data objects should be identified, alternative data organizations should be considered, and the impact of data modeling on software design should be evaluated.
2. *All data structures and the operations to be performed on each should be identified.* The design of an efficient data structure must take the operations to be performed on the data structure into account. The attributes and operations encapsulated within a class satisfy this principle.
3. *A mechanism for defining the content of each data object should be established and used to define both data and the operations applied to it.* Class diagrams

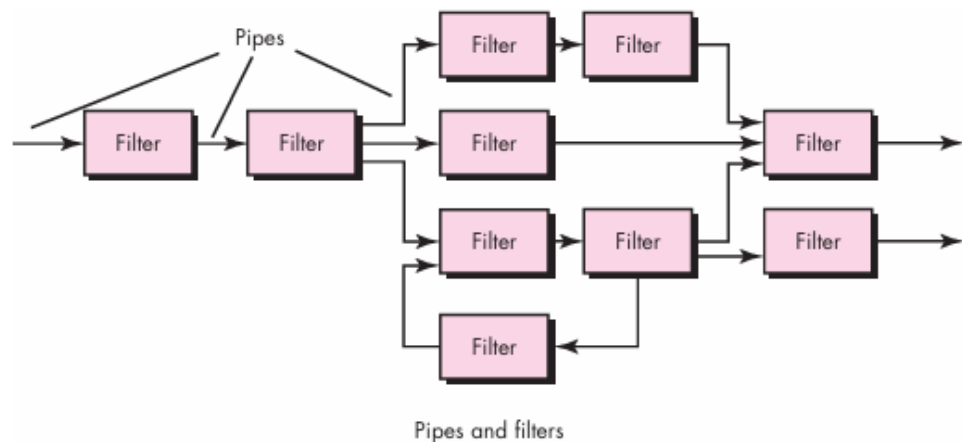
	<p>(Chapter 8) define the data items (attributes) contained within a class and the processing (operations) that are applied to these data items.</p> <ol style="list-style-type: none"> <li>4. <i>Low-level data design decisions should be deferred until late in the design process.</i> A process of stepwise refinement may be used for the design of data. That is, overall data organization may be defined during requirements analysis, refined during data design work, and specified in detail during component-level design.</li> <li>5. <i>The representation of a data structure should be known only to those modules that must make direct use of the data contained within the structure.</i> The concept of information hiding and the related concept of coupling (Chapter 9) provide important insight into the quality of a software design.</li> <li>6. <i>A library of useful data structures and the operations that may be applied to them should be developed.</i> A class library achieves this.</li> <li>7. <i>A software design and programming language should support the specification and realization of abstract data types.</i> The implementation of a sophisticated data structure can be made exceedingly difficult if no means for direct specification of the structure exists in the programming language chosen for implementation.</li> </ol> <p>These principles form a basis for a component-level data design approach that can be integrated into both the analysis and design activities.</p>			
4(c)	Discuss briefly the taxonomy of architectural styles.	10M	L2	C215.3
	<p>Although millions of computer-based systems have been created over the past 60 years, the vast majority can be categorized into one of a relatively small number of architectural styles:</p> <p><b>Data-centered architectures.</b> A data store (e.g., a file or database) resides at the center of this architecture and is accessed frequently by other components that update, add, delete, or otherwise modify data within the store. Figure 9.1 illustrates a typical data-centered style. Client software accesses a central repository. In some cases the data repository is passive. That is, client software accesses the data independent of any changes to the data or the actions of other client software. A variation on this approach transforms the repository into a “blackboard”</p>			



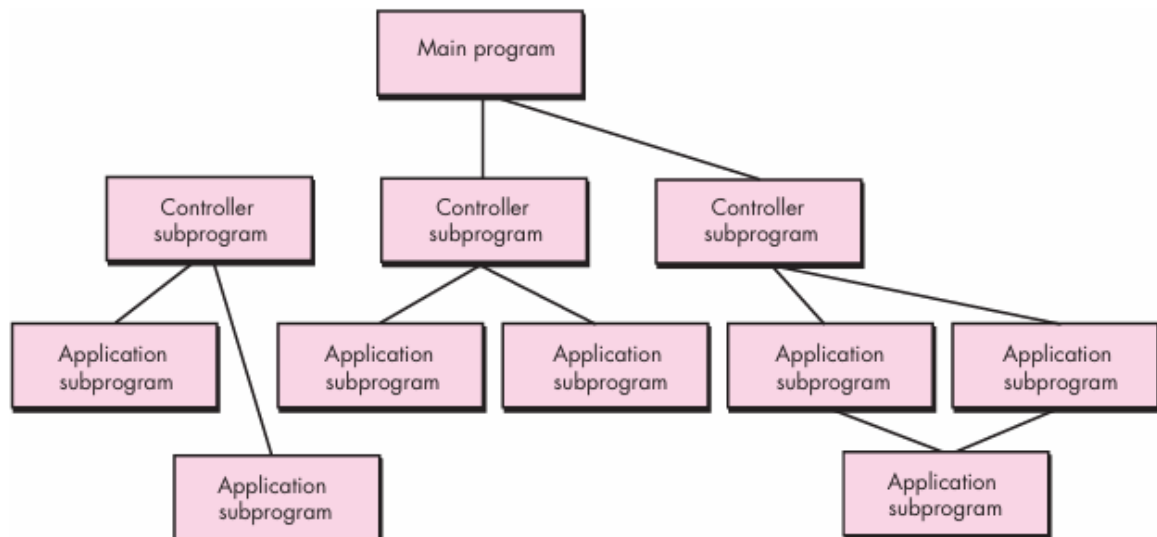
**FIGURE 9.1****Data-centered architecture**

that sends notifications to client software when data of interest to the client changes. Data-centered architectures promote integrability [Bas03]. That is, existing components can be changed and new client components added to the architecture without concern about other clients (because the client components operate independently). In addition, data can be passed among clients using the black board mechanism (i.e., the blackboard component serves to coordinate the transfer of information between clients). Client components independently execute processes. Data-flow architectures. This architecture is applied when input data are to be transformed through a series of computational or manipulative components into output data. A pipe-and-filter pattern (Figure 9.2) has a set of components, called filters, connected by pipes that transmit data from one component to the next. Each filter works independently of those components upstream and downstream, is designed to expect data input of a certain form, and produces data output (to the next filter) of a specified form. However, the filter does not require knowledge of the workings of its neighboring filters. If the data flow degenerates into a single line of transforms, it is termed batch sequential. This structure accepts a batch of data and then applies a series of sequential components (filters) to transform it. Call and return architectures. This architectural style enables you to achieve a program structure that is relatively easy to modify and scale. A number of substyles [Bas03] exist within this category:

- Main program/subprogram architectures. This classic program structure decomposes function into a control hierarchy where a “main” program

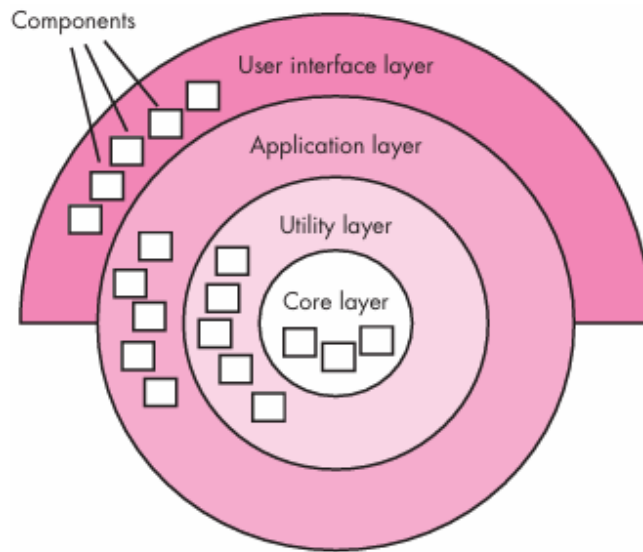
**FIGURE 9.2**Data-flow  
architecture**FIGURE 9.3**

Main program/subprogram architecture



invokes a number of program components that in turn may invoke still other components. Figure 9.3 illustrates an architecture of this type. • Remote procedure call architectures. The components of a main program/subprogram architecture are distributed across multiple computers on a network. Object-oriented architectures. The components of a system encapsulate data and the operations that must be applied to manipulate the data. Communication and coordination between components are accomplished via message passing.

**FIGURE 9.4**  
Layered  
architecture



Layered architectures. The basic structure of a layered architecture is illustrated in Figure 9.4. A number of different layers are defined, each accomplishing operations that progressively become closer to the machine instruction set. At the outer layer, components service user interface operations. At the inner layer, components perform operating system interfacing. Intermediate layers provide utility services and application software functions. These architectural styles are only a small subset of those available. Once requirements engineering uncovers the characteristics and constraints of the system to be built, the architectural style and/or combination of patterns that best fits those characteristics and constraints can be chosen. In many cases, more than one pattern might be appropriate and alternative architectural styles can be designed and evaluated. For example, a layered style (appropriate for most systems) can be combined with a data-centered architecture in many database applications.

Q. No	Question (s)	Marks	BL	CO
<b>UNIT - IV</b>				
1(a)	Define software testing.	1M	L1	C215.6
	<b>Software testing</b> is a crucial process in the software development lifecycle that involves evaluating and verifying that a software application is free of bugs, meets the technical requirements set by its design and development, and satisfies user requirements efficiently and effectively			
1(b)	What is “driver” in the unit test environment?	1M	L2	C215.6
	A driver is nothing more than a “main program” that accepts test case data, passes such data to the component (to be tested), and prints relevant results.			
1(c)	Define “stub” in the unit test environment.	1M	L1	C215.6
	A stub or “dummy subprogram” uses the subordinate module’s interface, may do minimal data manipulation, prints verification of entry, and returns control to the module undergoing testing.			
1(d)	Define system testing.	1M	L1	C215.6
	System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all work to verify that system elements have been properly integrated and perform allocated functions.			

1(e)	Give the two possible outcomes of a validation test.	1M	L2	C215.6
	(i) The function or performance characteristic conforms to specification and is accepted (ii) a deviation from specification is uncovered and a deficiency list is created.			
2(a)	Compare alpha and beta testing.	3M	L4	C215.6
	Alpha Testing	Beta Testing		
	The alpha test is conducted at the developer's site by a customer. The software is used in a natural setting with the developer "looking over the shoulder" of the user and recording errors and usage problems. Alpha tests are conducted in a controlled environment.	The beta test is conducted at one or more customer sites by the end-user of the software. Unlike alpha testing, the developer is generally not present.		
2(b)	Differentiate verification and validation.	3M	L4	C215.6
	Verification	Validation		
	Verification is to check whether the software conforms to specifications.	Validation is to check whether software meets the customer expectations and requirements.		
	Verification is done by QA team to ensure that the software is as per the specifications in the SRS document.	Validation is carried out with the involvement of testing team.		
	Verification uses methods like inspections, reviews, walkthroughs, and Desk-checking etc.	Validation uses methods like black box (functional) testing, gray box testing, and white box (structural) testing etc.		
	It is human based checking of documents and files.	It is computer based execution of program.		
2(c)	List the different types of integration testing.	3M	L2	C215.6
	i) Top down integration testing ii) Bottom up integration testing iii) Regression testing iv) Smoke testing			
2(d)	What is a critical module and why should we identify it?	3M	L2	C215.6
	A critical module has one or more of the following characteristics: (1) addresses several software requirements, (2) has a high level of control, (3) is complex or error prone, or (4) has definite performance requirements. Critical modules should be tested as early as is possible.			
2(e)	What do you understand by regression testing?	3M	L2	C215.6
	In the context of an integration test strategy, regression testing is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects. Regression testing may be conducted manually, by re-executing a subset of all test cases or using automated capture/playback tools.			

3(a)	Discuss the software testing strategy for conventional software architectures.	5M	L3	C215.6
	<p><b>1. Requirements Analysis:</b></p> <ul style="list-style-type: none"> <li>• Understanding User Needs: Thoroughly analyze and understand user requirements, functional specifications, and acceptance criteria to establish a solid foundation for testing activities.</li> <li>• Requirement Traceability: Establish traceability between requirements and test cases to ensure comprehensive coverage and alignment with user needs.</li> </ul> <p><b>2. Test Planning and Design:</b></p> <ul style="list-style-type: none"> <li>• Test Plan Development: Develop a comprehensive test plan outlining the testing approach, scope, objectives, resources, schedule, and deliverables for each phase of the project.</li> <li>• Test Case Design: Design test cases based on requirements, use cases, and functional specifications to validate the software's behavior and functionality.</li> </ul> <p><b>3. Test Execution:</b></p> <ul style="list-style-type: none"> <li>• Test Execution Phases: Execute tests in a structured manner, including unit testing, integration testing, system testing, and acceptance testing, to verify different aspects of the software.</li> <li>• Test Data Preparation: Prepare relevant test data sets to ensure comprehensive coverage of various scenarios and conditions during test execution.</li> <li>• Test Environment Setup: Establish and configure test environments that mirror production environments to simulate real-world conditions accurately.</li> </ul> <p><b>4. Defect Management:</b></p> <ul style="list-style-type: none"> <li>• Defect Reporting: Document and report defects using a standardized defect tracking system, including detailed information such as severity, priority, steps to reproduce, and affected components.</li> <li>• Defect Analysis: Analyze root causes of defects to identify underlying issues in requirements, design, or implementation and implement corrective actions accordingly.</li> </ul> <p><b>5. Regression Testing:</b></p> <ul style="list-style-type: none"> <li>• Regression Test Suite: Develop and maintain a regression test suite consisting of reusable test cases to verify that new changes or enhancements do not introduce regressions or break existing functionality.</li> <li>• Automated Regression Testing: Automate regression tests where possible to increase efficiency, reduce manual effort, and ensure consistent test execution.</li> </ul>			
3(b)	Write short notes on Smoke testing.	5M	L2	C215.6
	<p>Smoke testing is an integration testing approach that is commonly used when product software is developed. It is designed as a pacing mechanism for time-critical projects, allowing the software team to assess the project on a frequent basis.</p> <p>In essence, the smoke-testing approach encompasses the following activities:</p> <ol style="list-style-type: none"> <li>1. Software components that have been translated into code are integrated into a build. A build includes all data files, libraries, reusable modules, and engineered components that are required to implement one or more product functions.</li> <li>2. A series of tests is designed to expose errors that will keep the build from properly performing its function. The intent should be to uncover “show stopper” errors that have the highest likelihood</li> </ol>			

	<p>of throwing the software project behind schedule.</p> <p>3. The build is integrated with other builds, and the entire product (in its current form) is smoke tested daily. The integration approach may be top down or bottom up.</p> <p>Smoke testing provides a number of benefits when it is applied on complex, time critical software projects:</p> <ul style="list-style-type: none"> <li>• <b>Integration risk is minimized.</b> Because smoke tests are conducted daily, incompatibilities and other show-stopper errors are uncovered early, thereby reducing the likelihood of serious schedule impact when errors are uncovered.</li> <li>• <b>The quality of the end product is improved.</b> Because the approach is construction (integration) oriented, smoke testing is likely to uncover functional errors as well as architectural and component-level design errors. If these errors are corrected early, better product quality will result.</li> <li>• <b>Error diagnosis and correction are simplified.</b> Like all integration testing approaches, errors uncovered during smoke testing are likely to be associated with “new software increments”—that is, the software that has just been added to the build(s) is a probable cause of a newly discovered error.</li> <li>• <b>Progress is easier to assess.</b> With each passing day, more of the software has been integrated and more has been demonstrated to work. This improves team morale and gives managers a good indication that progress is being made.</li> </ul>			
3(c)	Black box and white box testing – Compare.	5M	L3	C215.6

	<table><tr><th>Black Box Testing</th><th>White Box Testing</th></tr><tr><td>Black box testing is the Software testing method which is used to test the software without knowing the internal structure of code or program.</td><td>White box testing is the software testing method in which internal structure is being known to tester who is going to test the software.</td></tr><tr><td>This type of testing is carried out by testers.</td><td>Generally, this type of testing is carried out by software developers.</td></tr><tr><td>Implementation Knowledge is not required to carry out Black Box Testing.</td><td>Implementation Knowledge is required to carry out White Box Testing.</td></tr><tr><td>Programming Knowledge is not required to carry out Black Box Testing.</td><td>Programming Knowledge is required to carry out White Box Testing.</td></tr><tr><td>Testing is applicable on higher levels of testing like System Testing, Acceptance testing.</td><td>Testing is applicable on lower level of testing like Unit Testing, Integration testing.</td></tr><tr><td>Black box testing means functional test or external testing.</td><td>White box testing means structural test or interior testing.</td></tr><tr><td>In Black Box testing is primarily concentrate on the functionality of the system under test.</td><td>In White Box testing is primarily concentrate on the testing of program code of the system under test like code structure, branches, conditions, loops etc.</td></tr><tr><td>The main aim of this testing to check on what functionality is performing by the system under test.</td><td>The main aim of White Box testing to check on how System is performing.</td></tr><tr><td>Black Box testing can be started based on Requirement Specifications documents.</td><td>White Box testing can be started based on Detail Design documents.</td></tr><tr><td>The Functional testing, Behavior testing, Close box testing is carried out under Black Box testing, so there is no required of the programming knowledge.</td><td>The Structural testing, Logic testing, Path testing, Loop testing, Code coverage testing, Open box testing is carried out under White Box testing, so there is compulsory to know about programming knowledge.</td></tr></table>	Black Box Testing	White Box Testing	Black box testing is the Software testing method which is used to test the software without knowing the internal structure of code or program.	White box testing is the software testing method in which internal structure is being known to tester who is going to test the software.	This type of testing is carried out by testers.	Generally, this type of testing is carried out by software developers.	Implementation Knowledge is not required to carry out Black Box Testing.	Implementation Knowledge is required to carry out White Box Testing.	Programming Knowledge is not required to carry out Black Box Testing.	Programming Knowledge is required to carry out White Box Testing.	Testing is applicable on higher levels of testing like System Testing, Acceptance testing.	Testing is applicable on lower level of testing like Unit Testing, Integration testing.	Black box testing means functional test or external testing.	White box testing means structural test or interior testing.	In Black Box testing is primarily concentrate on the functionality of the system under test.	In White Box testing is primarily concentrate on the testing of program code of the system under test like code structure, branches, conditions, loops etc.	The main aim of this testing to check on what functionality is performing by the system under test.	The main aim of White Box testing to check on how System is performing.	Black Box testing can be started based on Requirement Specifications documents.	White Box testing can be started based on Detail Design documents.	The Functional testing, Behavior testing, Close box testing is carried out under Black Box testing, so there is no required of the programming knowledge.	The Structural testing, Logic testing, Path testing, Loop testing, Code coverage testing, Open box testing is carried out under White Box testing, so there is compulsory to know about programming knowledge.			
Black Box Testing	White Box Testing																									
Black box testing is the Software testing method which is used to test the software without knowing the internal structure of code or program.	White box testing is the software testing method in which internal structure is being known to tester who is going to test the software.																									
This type of testing is carried out by testers.	Generally, this type of testing is carried out by software developers.																									
Implementation Knowledge is not required to carry out Black Box Testing.	Implementation Knowledge is required to carry out White Box Testing.																									
Programming Knowledge is not required to carry out Black Box Testing.	Programming Knowledge is required to carry out White Box Testing.																									
Testing is applicable on higher levels of testing like System Testing, Acceptance testing.	Testing is applicable on lower level of testing like Unit Testing, Integration testing.																									
Black box testing means functional test or external testing.	White box testing means structural test or interior testing.																									
In Black Box testing is primarily concentrate on the functionality of the system under test.	In White Box testing is primarily concentrate on the testing of program code of the system under test like code structure, branches, conditions, loops etc.																									
The main aim of this testing to check on what functionality is performing by the system under test.	The main aim of White Box testing to check on how System is performing.																									
Black Box testing can be started based on Requirement Specifications documents.	White Box testing can be started based on Detail Design documents.																									
The Functional testing, Behavior testing, Close box testing is carried out under Black Box testing, so there is no required of the programming knowledge.	The Structural testing, Logic testing, Path testing, Loop testing, Code coverage testing, Open box testing is carried out under White Box testing, so there is compulsory to know about programming knowledge.																									
3(d)	Explain System testing in detail.	5M	L2	C215.6																						
	<p>System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all work to verify that system elements have been properly integrated and perform allocated functions.</p> <p><b>1.Recovery Testing</b> Recovery testing is a system test that forces the software to fail in a variety of ways and verifies that recovery is properly performed. If recovery is automatic (performed by the system itself).</p>																									

reinitialization, checkpointing mechanisms, data recovery, and restart are evaluated for correctness. If recovery requires human intervention, the mean-time-to-repair (MTTR) is evaluated to determine whether it is within acceptable limits.

## 2.Security Testing

Security testing attempts to verify that protection mechanisms built into a system will, in fact, protect it from improper penetration. During security testing, the tester plays the role(s) of the individual who desires to penetrate the system. The tester may attempt to acquire passwords through external clerical means; may attack the system with custom software designed to break down any defenses that have been constructed; may overwhelm the system, thereby denying service to others; may purposely cause system errors, hoping to penetrate during recovery; may browse through insecure data, hoping to find the key to system entry. Given enough time and resources, good security testing will ultimately penetrate a system. The role of the system designer is to make penetration cost more than the value of the information that will be obtained.

## 3.Stress Testing

Stress testing executes a system in a manner that demands resources in abnormal quantity, frequency, or volume. A variation of stress testing is a technique called sensitivity testing. In some situations (the most common occur in mathematical algorithms), a very small range of data contained within the bounds of valid data for a program may cause extreme and even erroneous processing or profound performance degradation. Sensitivity testing attempts to uncover data combinations within valid input classes that may cause instability or improper processing.

## 4.Performance Testing

Performance testing is designed to test the run-time performance of software within the context of an integrated system. Performance testing occurs throughout all steps in the testing process. Even at the unit level, the performance of an individual module may be assessed as tests are conducted. However, it is not until all system elements are fully integrated that the true performance of a system can be ascertained.

## 5.Deployment Testing

Deployment testing, sometimes called configuration testing, exercises the software in each environment in which it is to operate. In addition, deployment testing examines all installation procedures and specialized installation software (e.g., “installers”) that will be used by customers, and all documentation that will be used to introduce the software to end users.

3(e)	Discuss the product metrics landscape.	5M	L2	C215.4
	<p><b>Product metrics landscape:</b></p> <p>Various metrics formulated for products in the development process are listed below. Metrics for analysis model: These address various aspects of the analysis model such as system functionality, system size, and so on.</p> <ul style="list-style-type: none"> <li>• <b>Metrics for design model:</b> These allow software engineers to assess the quality of design and include architectural design metrics, component-level design metrics, and so on.</li> <li>• <b>Metrics for source code:</b> These assess source code complexity, maintainability, and other characteristics.</li> </ul>			



- **Metrics for testing:** These help to design efficient and effective test cases and also evaluate the effectiveness of testing.
- **Metrics for maintenance:** These assess the stability of the software product.

4(a) Unit Testing – Explain.

10M

L2

C215.6

Unit testing focuses verification effort on the smallest unit of software design—the software component or module. Using the component-level design description as a guide, important control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests and the errors those tests uncover is limited by the constrained scope established for unit testing. The unit test focuses on the internal processing logic and data structures within the boundaries of a component. This type of testing can be conducted in parallel for multiple components.

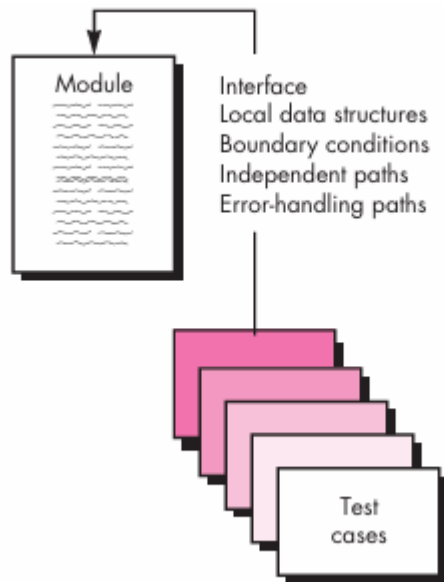


Fig.Unit test

#### Unit-test considerations:

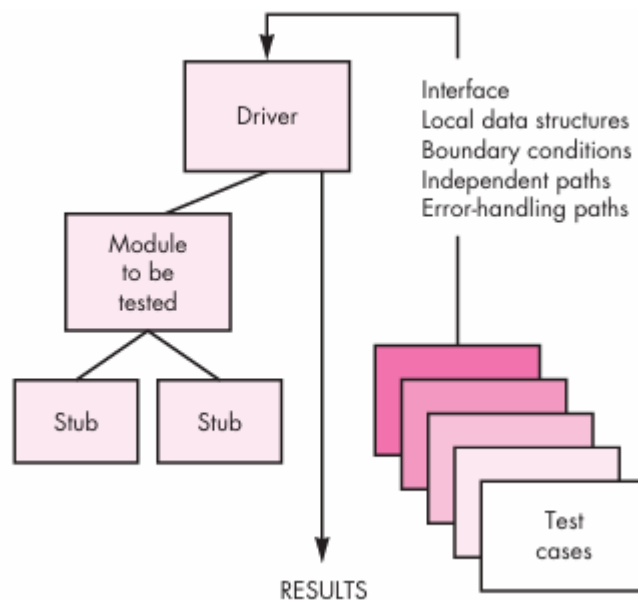
Unit tests are illustrated schematically in Figure. The module interface is tested to ensure that information properly flows into and out of the program unit under test. Local data structures are examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution. All independent paths through the control structure are exercised to ensure that all statements in a module have been executed at least once. Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing. And finally, all error-handling paths are tested. Data flow across a component interface is tested before any other testing is

initiated. If data do not enter and exit properly, all other tests are moot. In addition, local data structures should be exercised and the local impact on global data should be ascertained (if possible) during unit testing. Selective testing of execution paths is an essential task during the unit test. Test cases should be designed to uncover errors due to erroneous computations, incorrect

comparisons, or improper control flow. Boundary testing is one of the most important unit testing tasks. Software often fails at its boundaries. That is, errors often occur when the  $n$ th element of an  $n$ -dimensional array is processed, when the  $i^{\text{th}}$ -repetition of a loop with  $i$  passes is invoked, when the maximum or minimum allowable value is encountered. Test cases that exercise data structure, control flow, and data values just below, at, and just above maxima and minima are very likely to uncover errors.

### Unit-test procedures:

Unit testing is normally considered as an adjunct to the coding step. The design of unit tests can occur before coding begins or after source code has been generated. A review of design information provides guidance for establishing test cases that are likely to uncover errors in each of the categories discussed earlier. Each test case should be coupled with a set of expected results. Because a component is not a stand-alone program, driver and/or stub software must often be developed for each unit test. The unit test environment is illustrated in following Figure.



**Fig: Unit Test Environment**

In most applications a driver is nothing more than a “main program” that accepts test case data, passes such data to the component (to be tested), and prints relevant results. Stubs serve to replace modules that are subordinate (invoked by) the component to be tested. A stub or “dummy subprogram” uses the subordinate module’s interface, may do minimal data manipulation, prints verification of entry, and returns control to the module undergoing testing. Drivers and stubs represent testing “overhead.” That is, both are software that must be written (formal design is not commonly applied) but that is not delivered with the final software product. If drivers and stubs are kept simple, actual overhead is relatively low. Unfortunately, many components cannot be adequately unit tested with “simple” overhead software. In such cases, complete testing can be postponed until the integration test step (where drivers or stubs are also used). Unit testing is simplified when a component with high cohesion is designed. When only one function is addressed by a component, the number of test cases is reduced and errors can be more easily predicted and uncovered.

4(b)	Discuss in detail the art of debugging.	10M	L2	C215.4
	<p>Debugging occurs as a consequence of successful testing. That is, when a test case uncovers an error, debugging is the process that results in the removal of the error. Although debugging can and should be an orderly process, it is still very much an art.</p> <p><b>The Debugging Process</b></p> <p>The debugging process will usually have one of two outcomes:</p> <ol style="list-style-type: none"> <li>(1) the cause will be found and corrected or</li> <li>(2) the cause will not be found.</li> </ol> <p>In the latter case, the person performing debugging may suspect a cause, design a test case to help validate that suspicion, and work toward error correction in an iterative fashion.</p> <div data-bbox="323 705 1042 1419" data-label="Diagram"> <pre> graph TD     TC[Test Cases] --&gt; Comp[Computer]     Comp --&gt; Res[Results]     Res --&gt; Debug[Debugging]     Debug --&gt; IC[Identified causes]     IC --&gt; Cor[Corrections]     Cor --&gt; RT[Regression tests]     RT --&gt; TC     Debug --&gt; SC[Suspected causes]     SC --&gt; AT[Additional tests]     AT --&gt; TC   </pre> </div> <p><b>Fig. The debugging process</b></p> <p><b>Why is debugging so difficult?</b></p> <ol style="list-style-type: none"> <li>1. The symptom and the cause may be geographically remote.</li> <li>2. The symptom may disappear (temporarily) when another error is corrected.</li> <li>3. The symptom may actually be caused by non-errors (e.g., round-off inaccuracies).</li> <li>4. The symptom may be caused by human error that is not easily traced.</li> <li>5. The symptom may be a result of timing problems, rather than processing problems.</li> <li>6. It may be difficult to accurately reproduce input conditions (e.g., a real-time application in which input ordering is indeterminate).</li> <li>7. The symptom may be intermittent. This is particularly common in embedded systems that couple hardware and software inextricably.</li> <li>8. The symptom may be due to causes that are distributed across a number of tasks running on different processors.</li> </ol>			

### **Psychological Considerations**

Unfortunately, there appears to be some evidence that debugging prowess is an innate human trait. Some people are good at it and others aren't.

Debugging is one of the more frustrating parts of programming. It has elements of problem solving or brain teasers, coupled with the annoying recognition that you have made a mistake. Heightened anxiety and the unwillingness to accept the possibility of errors increases the task difficulty. Fortunately, there is a great sigh of relief and a lessening of tension when the bug is ultimately corrected.

### **Debugging Strategies**

In general, three debugging strategies have been proposed:

- (1) brute force,
- (2) backtracking, and
- (3) cause elimination.

Each of these strategies can be conducted manually, but modern debugging tools can make the process much more effective.

The **brute force category** of debugging is probably the most common and least efficient method for isolating the cause of a software error. Using a “let the computer find the error” philosophy, memory dumps are taken, run-time traces are invoked, and the program is loaded with output statements. You hope that somewhere in the morass of information that is produced you'll find a clue that can lead to the cause of an error.

**Backtracking** is a fairly common debugging approach that can be used success fully in small programs. Beginning at the site where a symptom has been uncovered, the source code is traced backward (manually) until the cause is found.

The third approach to debugging—**cause elimination**—is manifested by induction or deduction and introduces the concept of binary partitioning. Data related to the error occurrence are organized to isolate potential causes. A “cause hypothesis” is devised and the aforementioned data are used to prove or disprove the hypothesis.

### **Automated debugging:**


Many new approaches have been proposed and many commercial debugging environments are available. Integrated development environments (IDEs) provide a way to capture some of the language specific predetermined errors (e.g., missing end-of-statement characters, undefined variables, and so on) without requiring compilation.” A wide variety of debugging compilers, dynamic debugging aids (“tracers”), automatic test-case generators, and cross-reference mapping tools are available. However, tools are not a substitute for careful evaluation based on a complete design model and clear source code.

### **The people factor:**

Any discussion of debugging approaches and tools is incomplete without mention of a powerful ally—other people! A fresh viewpoint, un clouded by hours of frustration, can do wonders.

	<p><b>Correcting the Error:</b></p> <p>Once a bug has been found, it must be corrected. But, as we have already noted, the correction of a bug can introduce other errors and therefore do more harm than good.</p> <p>Van Vleck suggests three simple questions that you should ask before making the “correction” that removes the cause of a bug:</p> <ol style="list-style-type: none"> <li>1. <b>Is the cause of the bug reproduced in another part of the program?</b> In many situations, a program defect is caused by an erroneous pattern of logic that may be reproduced elsewhere. Explicit consideration of the logical pattern may result in the discovery of other errors.</li> <li>2. <b>What “next bug” might be introduced by the fix I’m about to make?</b> Before the correction is made, the source code (or, better, the design) should be evaluated to assess coupling of logic and data structures. If the correction is to be made in a highly coupled section of the program, special care must be taken when any change is made.</li> <li>3. <b>What could we have done to prevent this bug in the first place?</b> This question is the first step toward establishing a statistical software quality assurance approach. If you correct the process as well as the product, the bug will be removed from the current program and may be eliminated from all future programs.</li> </ol>
4(c)	<p>Explain in detail the software quality factors.</p> <p><b>10M</b>    L2    C215.4</p>
	<p>Software quality can be defined as: An effective software process applied in a manner that creates a useful product that provides measurable value for those who produce it and those who use it.</p> <p><b><u>Garvin’s Quality Dimensions</u></b></p> <p>David Garvin suggests that quality should be considered by taking a multidimensional viewpoint that begins with an assessment of conformance and terminates with a transcendental (aesthetic) view.</p> <p><b>Performance quality:</b> Does the software deliver all content, functions, and features that are specified as part of the requirements model in a way that provides value to the end user?</p> <p><b>Feature quality:</b> Does the software provide features that surprise and delight first-time end users?</p> <p><b>Reliability:</b> Does the software deliver all features and capability without failure? Is it available when it is needed? Does it deliver functionality that is error-free?</p> <p><b>Conformance:</b> Does the software conform to local and external software standards that are relevant to the application? Does it conform to de facto design and coding conventions?</p> <p><b>Durability:</b> Can the software be maintained (changed) or corrected (debugged) without the inadvertent generation of unintended side effects? Will changes cause the error rate or reliability to degrade with time?</p> <p><b>Serviceability:</b> Can the software be maintained (changed) or corrected (debugged) in an acceptably short time period? Can support staff acquire all information they need to make changes or correct defects?</p> <p><b>Aesthetics:</b> There’s no question that each of us has a different and very subjective vision of what is aesthetic. And yet, most of us would agree that an aesthetic entity has a certain elegance, a unique flow, and an obvious “presence” that are hard to quantify but are evident nonetheless. Aesthetic software has these characteristics.</p> <p><b>Perception:</b> In some situations, you have a set of prejudices that will influence your perception of quality. For example, if you are introduced to a software product that was built by a vendor who has produced poor quality in the past, your guard will be raised and your perception of the current software product quality might be influenced negatively. Similarly, if a vendor has an excellent</p>

	<p>reputation, you may perceive quality, even when it does not really exist.</p> <p><b>ISO 9126 Quality Factors</b></p> <p>The ISO 9126 standard was developed in an attempt to identify the key quality attributes for computer software. The standard identifies six key quality attributes:</p> <p><b>Functionality.</b> The degree to which the software satisfies stated needs as indicated by the following sub-attributes: suitability, accuracy, interoperability, compliance, and security.</p> <p><b>Reliability.</b> The amount of time that the software is available for use as indicated by the following sub-attributes: maturity, fault tolerance, recoverability.</p> <p><b>Usability.</b> The degree to which the software is easy to use as indicated by the following sub-attributes: understandability, learnability, operability.</p> <p><b>Efficiency.</b> The degree to which the software makes optimal use of system resources as indicated by the following sub-attributes: time behavior, resource behavior.</p> <p><b>Maintainability.</b> The ease with which repair may be made to the software as indicated by the following sub-attributes: analyzability, changeability, stability, testability.</p> <p><b>Portability.</b> The ease with which the software can be transposed from one environment to another as indicated by the following sub-attributes: adaptability, installability, conformance, replaceability.</p>			
Q. No	Question (s)	Marks	BL	CO
	<b>UNIT-V</b>			
1(a)	What are the two characteristics of software risks?	1M	L2	C215.5
	<ul style="list-style-type: none"> <li>• uncertainty—the risk may or may not happen; that is, there are no 100 percent probable risks</li> <li>• loss—if the risk becomes a reality, unwanted consequences or losses will occur.</li> </ul>			
1(b)	Define risk projection.	1M	L1	C215.5
	Risk projection, also called risk estimation, attempts to rate each risk in two ways— (1) the likelihood or probability that the risk is real and (2) the consequences of the problems associated with the risk, should it occur.			
1(c)	Define RMMM.	1M	L1	C215.5
	All risks that lie above the cutoff line should be managed. The RMMM contains a pointer into a risk mitigation, monitoring, and management plan or, alternatively, a collection of risk information sheets developed for all risks that lie above the cutoff.			
1(d)	Define software quality.	1M	L1	C215.5
	Software quality refers to the degree to which a software product meets specified requirements and satisfies customer expectations. It encompasses various attributes and characteristics that contribute to the overall effectiveness, efficiency, reliability, maintainability, and usability of the software.			
1(e)	Define quality assurance system.	1M	L1	C215.5
	Quality assurance system consists of the auditing and reporting functions that assess the effectiveness and completeness of quality control activities.			

2(a)	Differentiate Reactive and Proactive risk strategies.		3M	L3	C215.5																							
	<table><tr><th>Attribute</th><th>Proactive Risk Management</th><th>Reactive Risk Management</th></tr><tr><td>Approach</td><td>Anticipates and addresses risks before they occur</td><td>Addresses risks after they occur</td></tr><tr><td>Focus</td><td>Prevention and mitigation</td><td>Response and recovery</td></tr><tr><td>Timeframe</td><td>Long-term perspective</td><td>Short-term perspective</td></tr><tr><td>Proactivity</td><td>Actively identifies and manages risks</td><td>Reactively responds to risks</td></tr><tr><td>Planning</td><td>Emphasizes planning and risk assessment</td><td>May lack comprehensive planning</td></tr><tr><td>Cost</td><td>May require upfront investment but can save costs in the long run</td><td>May incur higher costs due to unplanned events</td></tr><tr><td>Effectiveness</td><td>Reduces the likelihood and impact of risks</td><td>Addresses risks after they have already occurred</td></tr></table>	Attribute	Proactive Risk Management	Reactive Risk Management	Approach	Anticipates and addresses risks before they occur	Addresses risks after they occur	Focus	Prevention and mitigation	Response and recovery	Timeframe	Long-term perspective	Short-term perspective	Proactivity	Actively identifies and manages risks	Reactively responds to risks	Planning	Emphasizes planning and risk assessment	May lack comprehensive planning	Cost	May require upfront investment but can save costs in the long run	May incur higher costs due to unplanned events	Effectiveness	Reduces the likelihood and impact of risks	Addresses risks after they have already occurred			
Attribute	Proactive Risk Management	Reactive Risk Management																										
Approach	Anticipates and addresses risks before they occur	Addresses risks after they occur																										
Focus	Prevention and mitigation	Response and recovery																										
Timeframe	Long-term perspective	Short-term perspective																										
Proactivity	Actively identifies and manages risks	Reactively responds to risks																										
Planning	Emphasizes planning and risk assessment	May lack comprehensive planning																										
Cost	May require upfront investment but can save costs in the long run	May incur higher costs due to unplanned events																										
Effectiveness	Reduces the likelihood and impact of risks	Addresses risks after they have already occurred																										
2(b)	What are Known risks, Predictable risks and Unpredictable risks?	3M	L2	C215.5																								
	<p><b>Known risks</b> are those that can be uncovered after careful evaluation of the project plan, the business and technical environment in which the project is being developed, and other reliable information sources (e.g., unrealistic delivery date, lack of documented requirements or software scope, poor development environment).</p> <p><b>Predictable risks</b> are extrapolated from past project experience (e.g., staff turnover, poor communication with the customer, dilution of staff effort as ongoing maintenance requests are serviced).</p> <p><b>Unpredictable risks</b> are the joker in the deck. They can and do occur, but they are extremely difficult to identify in advance.</p>																											
2(c)	List down the four risk projection steps.	3M	L2	C215.5																								
	<ol style="list-style-type: none"><li>1. Establish a scale that reflects the perceived likelihood of a risk.</li><li>2. Delineate the consequences of the risk.</li><li>3. Estimate the impact of the risk on the project and the product.</li><li>4. Assess the overall accuracy of the risk projection so that there will be no misunderstandings.</li></ol>																											
2(d)	Draw the reference model for technical reviews.	3M	L2	C215.5																								
																												
2(e)	What are the core steps of six sigma methodology?	3M	L2	C215.5																								
	<ul style="list-style-type: none"><li>• Define customer requirements and deliverables and project goals via well defined methods of customer communication.</li><li>• Measure the existing process and its output to determine current quality performance</li></ul>																											

	(collect defect metrics). <ul style="list-style-type: none"> <li>Analyze defect metrics and determine the vital few causes.</li> </ul>			
<b>3(a)</b>	Discuss Risk Components and Drivers.	<b>5M</b>	L2	C215.5
	<p>The U.S. Air Force has published a pamphlet that contains excellent guide lines for software risk identification and abatement. The Air Force approach requires that the project manager identify the risk drivers that affect software risk components— performance, cost, support, and schedule. In the context of this discussion, the risk components are defined in the following manner:</p> <ul style="list-style-type: none"> <li>Performance risk—the degree of uncertainty that the product will meet its requirements and be fit for its intended use.</li> <li>Cost risk—the degree of uncertainty that the project budget will be maintained.</li> <li>Support risk—the degree of uncertainty that the resultant software will be easy to correct, adapt, and enhance.</li> <li>Schedule risk—the degree of uncertainty that the project schedule will be maintained and that the product will be delivered on time.</li> </ul> <p>The impact of each risk driver on the risk component is divided into one of four impact categories—negligible, marginal, critical, or catastrophic.</p>			
<b>3(b)</b>	How do we assess the consequences of a risk?	<b>5M</b>	L2	C215.5
	<p>We can apply the following steps to determine the overall consequences of a risk:</p> <ol style="list-style-type: none"> <li>(1) determine the average probability of occurrence value for each risk component;</li> <li>(2) using Figure 28.1, determine the impact for each component based on the criteria shown, and</li> <li>(3) complete the risk table and analyze the results.</li> </ol> <p>The overall risk exposure RE is determined using the following relationship :</p> $RE = P \times C$ <p>where P is the probability of occurrence for a risk, and C is the cost to the project should the risk occur.</p> <p>For example, assume that the software team defines a project risk in the following manner:</p> <p><b>Risk identification.</b> Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.</p> <p><b>Risk probability.</b> 80 percent (likely).</p> <p><b>Risk impact.</b> Sixty reusable software components were planned. If only 70 percent can be used, 18 components would have to be developed from scratch (in addition to other custom software that has been scheduled for development). Since the average component is 100 LOC and local data indicate that the software engineering cost for each LOC is \$14.00, the overall cost (impact) to develop the components would be <math>18 \times 100 \times 14 = \\$25,200</math>.</p> <p><b>Risk exposure.</b> <math>RE = 0.80 \times 25,200 \sim \\$20,200</math>.</p> <p>Risk exposure can be computed for each risk in the risk table, once an estimate of the cost of the risk is made. The total risk exposure for all risks (above the cutoff in the risk table) can provide a means for adjusting the final cost estimate for a project. It can also be used to predict the probable increase in staff resources required at various points during the project schedule.</p>			
<b>3(c)</b>	Explain Risk refinement.	<b>5M</b>	L2	C215.5
	<p>During early stages of project planning, a risk may be stated quite generally. As time passes and more is learned about the project and the risk, it may be possible to refine the risk into a set of more detailed risks, each somewhat easier to mitigate, monitor, and manage. One way to do this is to represent the risk in condition-transition-consequence (CTC) format. That is, the risk is stated in</p>			



the following form:

Given that then there is concern that (possibly) <consequence>

Using the CTC format for the reuse risk, you could write: Given that all reusable software components must conform to specific design standards and that some do not conform, then there is concern that (possibly) only 70 percent of the planned reusable modules may actually be integrated into the as-built system, resulting in the need to custom engineer the remaining 30 percent of components.

This general condition can be refined in the following manner:

**Subcondition 1.** Certain reusable components were developed by a third party with no knowledge of internal design standards.

**Subcondition 2.** The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.

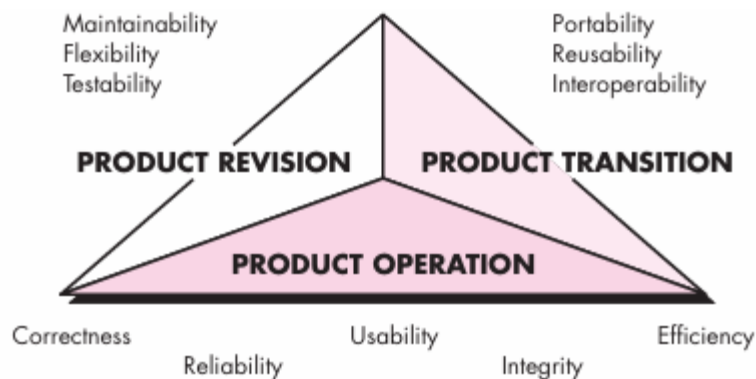
**Subcondition 3.** Certain reusable components have been implemented in a language that is not supported on the target environment.

The consequences associated with these refined subconditions remain the same (i.e., 30 percent of software components must be custom engineered), but the refinement helps to isolate the underlying risks and might lead to easier analysis and response.

3(d)	How does McCall categorize the factors that affect software quality?	5M	L2	C215.5
------	--	----	----	--------

### McCall's Quality Factors

McCall, Richards, and Walters propose a useful categorization of factors that affect software quality. These software quality factors, shown in following Figure, focus on three important aspects of a software product: its operational characteristics, its ability to undergo change, and its adaptability to new environments.



**Fig: McCall's Software quality factors**

**Correctness:** The extent to which a program satisfies its specification and fulfills the customer's mission objectives.

**Reliability:** The extent to which a program can be expected to perform its intended function with required precision.

**Efficiency:** The amount of computing resources and code required by a program to perform its

	<p>function.</p> <p><b>Integrity:</b> Extent to which access to software or data by unauthorized persons can be controlled.</p> <p><b>Usability:</b> Effort required to learn, operate, prepare input for, and interpret output of a program</p> <p><b>Maintainability:</b> Effort required to locate and fix an error in a program.</p> <p><b>Flexibility:</b> Effort required to modify an operational program.</p> <p><b>Testability:</b> Effort required to test a program to ensure that it performs its intended function.</p> <p><b>Portability:</b> Effort required to transfer the program from one hardware and/or software system environment to another.</p> <p><b>Reusability:</b> Extent to which a program [or parts of a program] can be reused in other applications—related to the packaging and scope of the functions that the program performs.</p> <p><b>Interoperability:</b> Effort required to couple one system to another.</p>			
3(e)	What are the roles of an SQA group in achieving a high quality end product?	5M	L2	C215.5
	<p>The charter of the SQA group is to assist the software team in achieving a high quality end product. The Software Engineering Institute recommends a set of SQA actions that address quality assurance planning, oversight, record keeping, analysis, and reporting. These actions are performed (or facilitated) by an independent SQA group that:</p> <p><b>Prepares an SQA plan for a project.</b> The plan is developed as part of project planning and is reviewed by all stakeholders. Quality assurance actions performed by the software engineering team and the SQA group are governed by the plan. The plan identifies evaluations to be performed, audits and reviews to be conducted, standards that are applicable to the project, procedures for error reporting and tracking, work products that are produced by the SQA group, and feedback that will be provided to the software team.</p> <p><b>Participates in the development of the project's software process description.</b> The software team selects a process for the work to be performed. The SQA group reviews the process description for compliance with organizational policy, internal software standards, externally imposed standards (e.g., ISO-9001), and other parts of the software project plan.</p> <p><b>Reviews software engineering activities to verify compliance with the defined software process.</b> The SQA group identifies, documents, and tracks deviations from the process and verifies that corrections have been made.</p> <p><b>Audits designated software work products to verify compliance with those defined as part of the software process.</b> The SQA group reviews selected work products; identifies, documents, and tracks deviations; verifies that corrections have been made; and periodically reports the results of its work to the project manager.</p> <p><b>Ensures that deviations in software work and work products are documented and handled according to a documented procedure.</b> Deviations may be encountered in the project plan, process description, applicable standards, or software engineering work products.</p> <p><b>Records any noncompliance and reports to senior management.</b> Noncompliance items are tracked until they are resolved.</p>			
4(a)	Explain in detail Risk Mitigation, Monitoring and Management.	10M	L2	C215.5
	<p>All of the risk analysis activities presented to this point have a single goal—to assist the project team in developing a strategy for dealing with risk. An effective strategy must consider three issues: risk avoidance, risk monitoring, and risk management and contingency planning.</p> <p>If a software team adopts a proactive approach to risk, avoidance is always the best strategy. This is achieved by developing a plan for risk mitigation. For example, assume that high staff turnover</p>			

is noted as a project risk r1. Based on past history and management intuition, the likelihood l1 of high turnover is estimated to be 0.70 (70 percent, rather high) and the impact x1 is projected as critical. That is, high turnover will have a critical impact on project cost and schedule. To mitigate this risk, you would develop a strategy for reducing turnover.

Among the possible steps to be taken are:

- Meet with current staff to determine causes for turnover (e.g., poor working conditions, low pay, competitive job market).
- Mitigate those causes that are under your control before the project starts.
- Once the project commences, assume turnover will occur and develop techniques to ensure continuity when people leave.
- Organize project teams so that information about each development activity is widely dispersed
- Define work product standards and establish mechanisms to be sure that all models and documents are developed in a timely manner.
- Conduct peer reviews of all work (so that more than one person is “up to speed”).
- Assign a backup staff member for every critical technologist.

If RE for a specific risk is less than the cost of risk mitigation, don't try to mitigate the risk but continue to monitor it. As the project proceeds, risk-monitoring activities commence. The project manager monitors factors that may provide an indication of whether the risk is becoming more or less likely. In the case of high staff turnover, the general attitude of team members based on project pressures, the degree to which the team has jelled, inter personal relationships among team members, potential problems with compensation and benefits, and the availability of jobs within the company and outside it are all monitored.

In addition to monitoring these factors, a project manager should monitor the effectiveness of risk mitigation steps. For example, a risk mitigation step noted here called for the definition of work product standards and mechanisms to be sure that work products are developed in a timely manner. This is one mechanism for ensuring continuity, should a critical individual leave the project. The project manager should monitor work products carefully to ensure that each can stand on its own and that each imparts information that would be necessary if a newcomer were forced to join the software team somewhere in the middle of the project.

**Risk management and contingency planning** assumes that mitigation efforts have failed and that the risk has become a reality. Continuing the example, the project is well under way and a number of people announce that they will be leaving. If the mitigation strategy has been followed, backup is available, information is documented, and knowledge has been dispersed across the team. In addition, you can temporarily refocus resources (and readjust the project schedule) to those functions that are fully staffed, enabling newcomers who must be added to the team to “get upto speed.” Those individuals who are leaving are asked to stop all work and spend their last weeks in “knowledge transfer mode.” This might include video-based knowledge capture, the development of “commentary documents or Wikis,” and/or meeting with other team members who will remain on the project.

It is important to note that risk mitigation, monitoring, and management (RMMM) steps incur additional project cost. For example, spending the time to back up every critical technologist costs money. Part of risk management, therefore, is to evaluate when the benefits accrued by the RMMM steps are outweighed by the costs associated with implementing them. In essence, you

	<p>perform a classic cost-benefit analysis. If risk aversion steps for high turnover will increase both project cost and duration by an estimated 15 percent, but the predominant cost factor is “backup,” management may decide not to implement this step. On the other hand, if the risk aversion steps are projected to increase costs by 5 percent and duration by only 3 percent, management will likely put all into place.</p> <p>For a large project, 30 or 40 risks may be identified. If between three and seven risk management steps are identified for each, risk management may become a project in itself! For this reason, you should adapt the Pareto 80–20 rule to software risk. Experience indicates that 80 percent of the overall project risk (i.e., 80 percent of the potential for project failure) can be accounted for by only 20 percent of the identified risks. The work performed during earlier risk analysis steps will help you to determine which of the risks reside in that 20 percent (e.g., risks that lead to the highest risk exposure). For this reason, some of the risks identified, assessed, and projected may not make it into the RMMM plan—they don’t fall into the critical 20 percent (the risks with highest project priority).</p> <p>Risk is not limited to the software project itself. Risks can occur after the software has been successfully developed and delivered to the customer. These risks are typically associated with the consequences of software failure in the field.</p> <p><b>Software safety and hazard analysis</b> are software quality assurance activities that focus on the identification and assessment of potential hazards that may affect software negatively and cause an entire system to fail. If hazards can be identified early in the software engineering process, software design features can be specified that will either eliminate or control potential hazards.</p>			
<b>4(b)</b>	Explain Formal Technical Reviews in detail.	<b>10M</b>	L2	C215.5
	<p>A formal technical review (FTR) is a software quality control activity performed by software engineers (and others). The objectives of an FTR are: (1) to uncover errors in function, logic, or implementation for any representation of the software; (2) to verify that the software under review meets its requirements; (3) to ensure that the software has been represented according to predefined standards; (4) to achieve software that is developed in a uniform manner; and (5) to make projects more manageable. In addition, the FTR serves as a training ground, enabling junior engineers to observe different approaches to software analysis, design, and implementation. The FTR also serves to promote backup and continuity because a number of people become familiar with parts of the software that they may not have otherwise seen. The FTR is actually a class of reviews that includes walkthroughs and inspections. Each FTR is conducted as a meeting and will be successful only if it is properly planned, controlled, and attended. In the sections that follow, guidelines similar to those for a walkthrough are presented as a representative formal technical review.</p> <p><b>The Review Meeting</b></p> <p>Regardless of the FTR format that is chosen, every review meeting should abide by the following constraints:</p> <ul style="list-style-type: none"> <li>• Between three and five people (typically) should be involved in the review.</li> <li>• Advance preparation should occur but should require no more than two hours of work for each person.</li> <li>• The duration of the review meeting should be less than two hours.</li> </ul>			

Given these constraints, it should be obvious that an FTR focuses on a specific (and small) part of the overall software.

The focus of the FTR is on a work product (e.g., a portion of a requirements model, a detailed component design, source code for a component). The individual who has developed the work product—the producer—informs the project leader that the work product is complete and that a review is required. The project leader contacts a review leader, who evaluates the product for readiness, generates copies of product materials, and distributes them to two or three reviewers for advance preparation. Each reviewer is expected to spend between one and two hours reviewing the product, making notes, and otherwise becoming familiar with the work. Concurrently, the review leader also reviews the product and establishes an agenda for the review meeting, which is typically scheduled for the next day.

The review meeting is attended by the review leader, all reviewers, and the producer. One of the reviewers takes on the role of a recorder, that is, the individual who records (in writing) all important issues raised during the review. The FTR begins with an introduction of the agenda and a brief introduction by the producer. The producer then proceeds to “walk through” the work product, explaining the material, while reviewers raise issues based on their advance preparation. When valid problems or errors are discovered, the recorder notes each.

At the end of the review, all attendees of the FTR must decide whether to: (1) accept the product without further modification, (2) reject the product due to severe errors (once corrected, another review must be performed), or (3) accept the product provisionally (minor errors have been encountered and must be corrected, but no additional review will be required). After the decision is made, all FTR attendees complete a sign-off, indicating their participation in the review and their concurrence with the review team’s findings.

### **Review Reporting and Record Keeping**

During the FTR, a reviewer (the recorder) actively records all issues that have been raised. These are summarized at the end of the review meeting, and a review issues list is produced. In addition, a formal technical review summary report is completed. A review summary report answers three questions:

1. What was reviewed?
2. Who reviewed it?
3. What were the findings and conclusions?

The review summary report is a single page form (with possible attachments). It becomes part of the project historical record and may be distributed to the project leader and other interested parties.

The review issues list serves two purposes: (1) to identify problem areas within the product and (2) to serve as an action item checklist that guides the producer as corrections are made. An issues list is normally attached to the summary report.

You should establish a follow-up procedure to ensure that items on the issues list have been properly corrected. Unless this is done, it is possible that issues raised can “fall between the cracks.” One approach is to assign the responsibility for follow-up to the review leader.

	<p><b>Review Guidelines</b></p> <ol style="list-style-type: none"> <li>1. Review the product, not the producer.</li> <li>2. Set an agenda and maintain it.</li> <li>3. Limit debate and rebuttal.</li> <li>4. Enunciate problem areas, but don't attempt to solve every problem noted.</li> <li>5. Take written notes.</li> <li>6. Limit the number of participants and insist upon advance preparation.</li> <li>7. Develop a checklist for each product that is likely to be reviewed.</li> <li>8. Allocate resources and schedule time for FTRs.</li> <li>9. Conduct meaningful training for all reviewers.</li> <li>10. Review your early reviews.</li> </ol>
4(c)	<p>What are the Measures of software reliability and availability? Also write briefly about software safety.</p> <p><b>10M</b>      L2      C215.5</p>
	<p><b>SOFTWARE RELIABILITY</b></p> <p>Software reliability is defined in statistical terms as "the probability of failure-free operation of a computer program in a specified environment for a specified time".</p> <p><b>Measures of Reliability and Availability:</b></p> <p>Most hardware-related reliability models are predicated on failure due to wear rather than failure due to design defects. In hardware, failures due to physical wear (e.g., the effects of temperature, corrosion, shock) are more likely than a design-related failure. Unfortunately, the opposite is true for software. In fact, all software failures can be traced to design or implementation problems; wear does not enter into the picture.</p> <p>A simple measure of reliability is meantime-between-failure (MTBF), where</p> $\text{MTBF} = \text{MTTF} + \text{MTTR}$ <p>The acronyms MTTF and MTTR are mean-time-to-failure and mean-time-to-repair, respectively.</p> <p>In addition to a reliability measure, we must develop a measure of availability. Software availability is the probability that a program is operating according to requirements at a given point in time and is defined as</p> $\text{Availability} = [\text{MTTF}/(\text{MTTF} + \text{MTTR})] \text{ 100\%}$ <p>The MTBF reliability measure is equally sensitive to MTTF and MTTR. The availability measure is somewhat more sensitive to MTTR, an indirect measure of the maintainability of software.</p> <p><b>Software Safety</b></p> <p>Software safety is a software quality assurance activity that focuses on the identification and assessment of potential hazards that may affect software negatively and cause an entire system to fail. If hazards can be identified early in the software engineering process, software design features can be specified that will either eliminate or control potential hazards.</p> <p>For example, some of the hazards associated with a computer-based cruise control for an automobile might be</p> <ul style="list-style-type: none"> <li>• causes uncontrolled acceleration that cannot be stopped</li> <li>• does not respond to depression of brake pedal (by turning off)</li> <li>• does not engage when switch is activated</li> </ul>

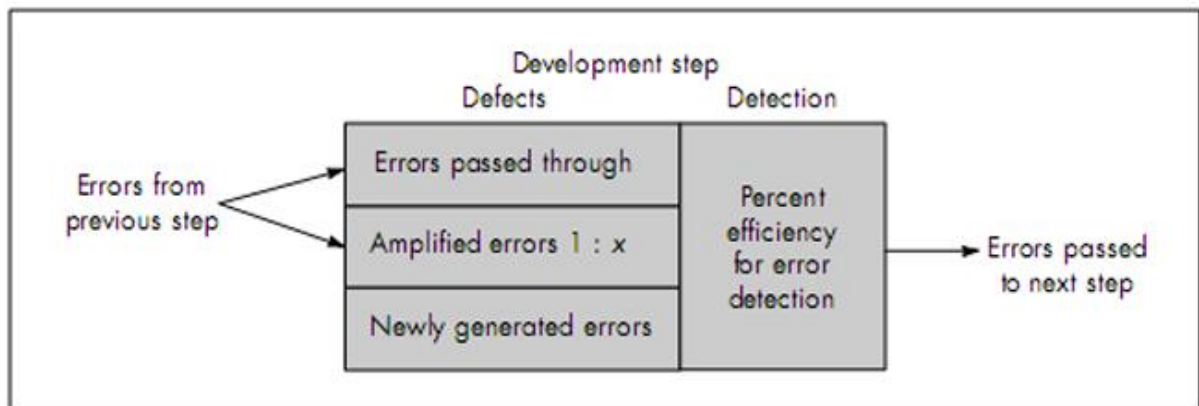
- slowly loses or gains speed

Once these system-level hazards are identified, analysis techniques are used to assign severity and probability of occurrence. To be effective, software must be analyzed in the context of the entire system. If a set of external environmental conditions are met (and only if they are met), the improper position of the mechanical device will cause a disastrous failure. Analysis techniques such as fault tree analysis, real-time logic, or petri net models can be used to predict the chain of events that can cause hazards and the probability that each of the events will occur to create the chain.

Once hazards are identified and analyzed, safety-related requirements can be specified for the software. That is, the specification can contain a list of undesirable events and the desired system responses to these events. The role of software in managing undesirable events is then indicated.

Although software reliability and software safety are closely related to one another, it is important to understand the subtle difference between them. Software reliability uses statistical analysis to determine the likelihood that a software failure will occur. However, the occurrence of a failure does not necessarily result in a hazard or mishap. Software safety examines the ways in which failures result in conditions that can lead to a mishap.

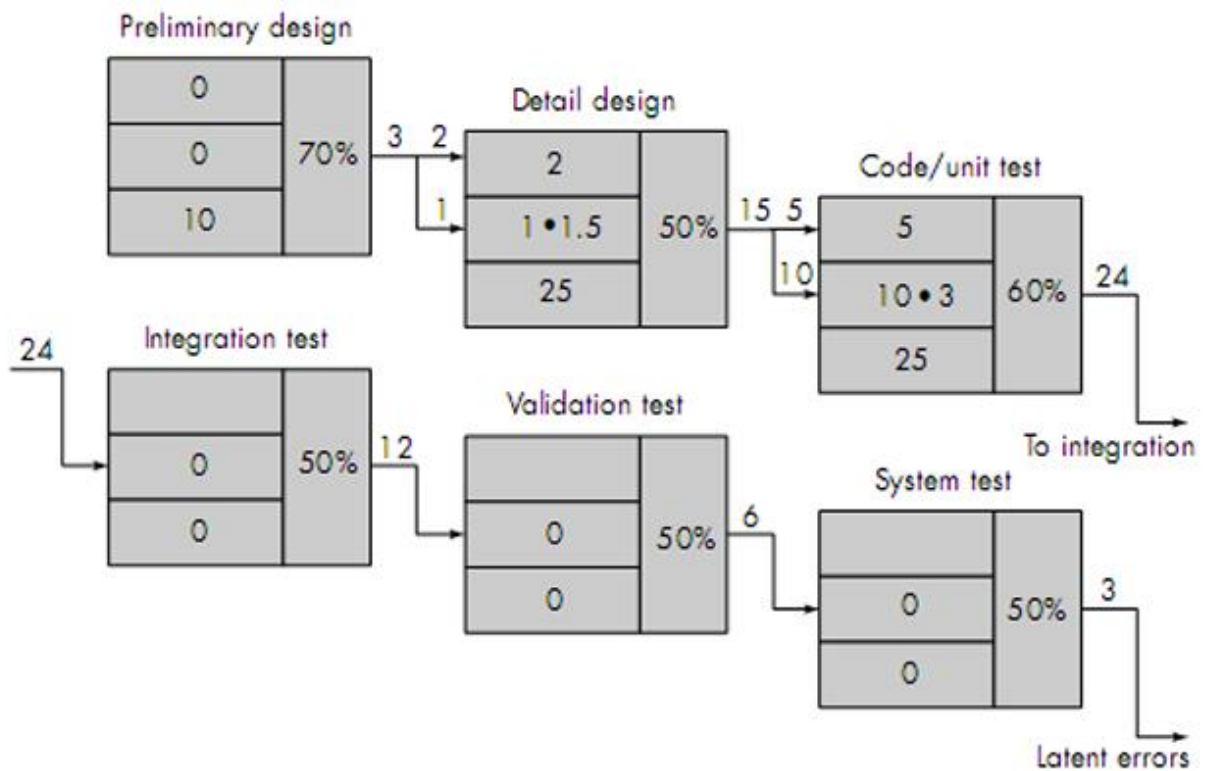
#### Defect Amplification and Removal:



**Fig:** A defect amplification model

A defect amplification model can be used to illustrate the generation and detection of errors during the preliminary design, detail design, and coding steps of the software engineering process. A box represents a software development step. During the step, errors may be inadvertently generated. Review may fail to uncover newly generated errors and errors from previous steps, resulting in some number of errors that are passed through. In some cases, errors passed through from previous steps are amplified (amplification factor,  $x$ ) by current work. The box subdivisions represent each of these characteristics and the percent of efficiency for detecting errors, a function of the thoroughness of the review.

### Defect amplification, reviews conducted



### Defect amplification, no reviews

