***THATHACOM** ee copyude aishvaryam*

Q1. What you mean by software engineering? software engineering as an engineering branch associated with the development of software product using well-defined scientific principles, methods and procedure. **Q2. Explain about the characteristics of a good software?** A goodsoftware must satisfy on the following grounds: • Operational • Transitional • Maintenance,/Operational :This tells us how well software works in operations. It can be measured on: • Budget • Usability • Efficiency • Correctness • Functionality • Dependability • Security • Safety, /Transitional :This aspect is important when the software is moved from one platform to another: • Portability • Interoperability • Reusability • Adaptability, /Maintenance. This aspect briefs about how well a software has the capabilities to maintain itself in the everchanging environment: • Modularity • Maintainability • Flexibility • Scalability **Q3. Explain about software process?** A software process is a collection of various activities. There are five generic process framework activities:, /• Communication: The software development starts with the communication between customer and developer., /• Planning: It consists of complete estimation, scheduling for project development and tracking., /•Modeling:Modeling consists of complete requirement analysis and the design of the project like algorithm, flowchart etc. The algorithm is the step-by-step solution of the problem and the flow chart shows a complete flow diagram of a program., /• Construction: Construction consists of code generation and the testing part.Coding part implements the design details using an appropriate programming language. Testing is to check whether the flow of coding is correct or not.Testing also check that the program provides desired output. • Deployment: Deployment step consists of delivering the product to the customer and take feedback from them.If the customer wants some corrections or demands for the additional capabilities, then the change is required for improvement in the quality of the software. **Q4. Explain umbrella activities?** The generic view of software engineering are complemented by a number of umbrella activities. Typical activities in this category include: • Software project tracking and control : Tracking and Control is the dual process of detecting when a project is drifting off-plan, and taking corrective action to bring the project back on track. • Formal technical reviews : This includes reviewing the techniques that has been used in the project. • Software quality assurance : This is very important to ensure the quality measurement of each part to ensure them. • Software configuration management :In software engineering, software configuration management (SCM or S/W CM) is the task of tracking and controlling changes in the software, part of the larger cross-disciplinary field of configuration management. • Document preparation and production : All the project planning and other activities should be hardly copied and the production get started here. • Reusability management : This includes the backing up of each part of the s/w project for the future work(eg: updation) • Measurement : This will include all the measurement of every aspects of the software project. • Risk management : Risk management is a series of steps that help a software team to understand and manage uncertainty. **Q5. Expn about layered structure of SE?** . Tools: This layer contains automated or semi-automated tools that offer support for the framework and the method each software engineering project will follow. • Method: This layer contains the methods, the technical knowledge and "how-tos" in order to develop software. • Process: This layer consists of the framework that must be established for the effective delivery of software. • Quality Focus: This layer is the fundamental layer for software engineering. As stated above it is of great importance to test the end product to see if it meets its specifications. Efficiency, usability, maintenance and reusability are some of the requirements that need to be met by new software. **Q6**. **Explain about SDLC?** SDLC is the acronym of Software Development Life Cycle.It is also called as Software Development Process.SDLC is a frame work defining tasks performed at each step in the software development process.A typical Software Development Life Cycle consists of the following stages stage 1: Planning and Requirement Analysis/preliminary investigation Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. Stage 2: Defining Requirements/software analysis Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an SRS (Software Requirement Specification) document which consists of all the product requirements to be designed and developed during the project life cycle. Stage 3: Designing the Product Architecture/software design SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.Eg: DFD,ER diagram etc... Stage 4: Building or Developing the Product/software coding In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle. Stage 5: Testing the Product/software testing This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS. Stage 6: Deployment in the Market and Maintenance/software maintananceOnce the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base. **Q7. What is prescriptive process model?** A prescriptive process model is a model that describes "how to do" according to a certain software process system.A prescriptive model prescribes how a new software system should be developed. Prescriptive models are used as guidelines or frameworks to organize and structure how software development activities should be performed, and in what order. **8. Give some examples of prescriptive process model?** • Waterfall model • V model • Incremental model • Evolutionary model • Prototyping • Spiral model • Win-Win Spiral model • Concurrent development **Q9. Explain about waterfall model?** The Waterfall Model was the first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases. Characteristics of waterfall model:- • It also known as Linear sequential model,Classic life cycle model. • Simplest of all models • Systematic sequential approach • Real project rarely follows • A working version is available only at the end • Difficult for customer to state all requirements• Requirement Gathering and analysis – All possible requirements of the system to be developed are captured in this phase and documented in a requirement. specification document. • System Design – 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. • Implementation – 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. • Integration and Testing – 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. • Deployment of system – Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market. • Maintenance – 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. **Q10. List the advantages and disadvantages of waterfall model?** Advantages: • Simple and easy to understand and use . Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process. • Phases are processed and completed one at a time. • Works well for smaller projects where requirements are very well understood. • Clearly defined stages. • Well understood milestones. • Easy to arrange tasks. • Process and results are well documented. Disadvantages: • No working software is produced until late during the life cycle. • High amounts of risk and uncertainty. • Not a good model for complex and object-oriented projects. • Poor model for long and ongoing projects. • Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model. • It is difficult to measure progress within stages. • Cannot accommodate changing requirements. • Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early. **Q11. Explain about spiral model?** Spiral model is one of the most important Software Development Life Cycle models, which provides support for Risk Handling. In its diagrammatic representation, it looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. Each loop of the spiral is called a Phase of the software development process. The exact number of phases needed to develop the product can be varied by the projectmanager depending upon the project risks. As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using spiral model. .. Objectives determination and identify alternative solutions: Requirements are gathered from the customers and the objectives are identified, elaborated and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant. • Identify and resolve Risks: During the second quadrant all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution is identified and the risks are resolved using the best possible strategy. At the end of this quadrant, Prototype is built for the best possible solution. • Develop next version of the Product: During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available. • Review and plan for the next Phase: In the fourth quadrant, the Customers evaluate the so far developed version of the software. In the end, planning for the next phase is started. **Q12. List the advantages and disadvantages of spiral model?** Advantages • Risk Handling: The projects with many unknown risks that occur as the development proceeds, in that case, Spiral Model is the best development model to follow due to the risk analysis and risk handling at every phase. • Good for large projects: It is recommended to use the Spiral Model in large and complex projects. • Flexibility in Requirements: Change requests in the Requirements at later phase can be incorporated accurately by using this model. • Customer Satisfaction: Customer can see the development of the product at the early phase of the software development and thus, they habituated with the system by using it before completion of the total product. Disadvantages: • Complex: The Spiral Model is much more complex than other SDLC models. • Expensive: Spiral Model is not suitable for small projects as it is expensive. • Too much dependable on Risk Analysis: The successful completion of the project is very much dependent on Risk Analysis. Without very highly experienced expertise, it is going to be a failure to develop a project using this model.• Difficulty in time management: As the number of phases is unknown at the start of the project, so time estimation is very difficult.

13. Explain about incremental model? Incremental Model is a process of software development where requirements are broken down into multiple standalone modules of software development cycle. Incremental development is done in steps from analysis design, implementation, testing/verification, maintenance.Each iteration passes through the requirements, design, coding and testing phases. And each subsequent release of the system adds function to the previous release until all designed functionality has been implemented. The system is put into production when the first increment is delivered. The first increment is often a core product where the basic requirements are addressed, and supplementary features are added in the next increments. Once the core product is analyzed by the client, there is plan development for the next increment. • Analysis:Requirement and specification of the software are collected • Design:Some high-end function are designed during this stage • Code:Coding of software is done during this stage • Test:Once the system is deployed, it goes through the testing phase **Q14. Characteristics of incremental model?** Analysis:Requirement and specification of the software are collected. Design:Some high-end function are designed during this stage. Code:Coding of software is done during this stage. Test:Once the system is deployed, it goes through the testing phase **Q15. What are the advantages and disadvantages of incremental model?** Advantages :• The software will be generated quickly during the software life cycle • It is flexible and less expensive • Throughout the development stages changes can be done • This model is less costly compared to others • A customer can respond to each building • Errors are easy to be identified Disadvantages :• It requires a good planning designing• Problems might cause due to system architecture as such not all requirements collected up front for the entire software lifecycle • Each iteration phase is rigid and does not overlap each other • Rectifying a problem in one unit requires correction in all the units and consumes a lot of time. **Q16. Explain about evolutionary prototyping model?** Evolutionary models are iterative type models.They allow to develop more complete versions of the software.It is software working model of limited functionality.Prototype methodology is defined as a Software Development model in which a prototype is built, test, and then reworked when needed until an acceptable prototype is achieved.First of all, we will develop the most visual aspect of the system. You present a portion of the system to the customer and continue to develop prototypes based on the feedback received. At some point, you and the customer agree that the prototype is “good enough” and release the prototype as the final product. Step 1: Requirements gathering and analysis. • A prototyping model starts with requirement analysis. In this phase, the requirements of the system are defined in detail. During the process, the users of the system are interviewed to know what is their expectation from the system. Step 2: Quick design. • The second phase is a preliminary design or a quick design. In this stage, a simple design of the system is created. However, it is not a complete design. It gives a brief idea of the system to the user. The quick design helps in developing the prototype. Step 3: Build a Prototype. • In this phase, an actual prototype is designed based on the information gathered from quick design. It is a small working model of the required system. Step 4: Initial user evaluation. • In this stage, the proposed system is presented to the client for an initial evaluation. It helps to find out the strength and weakness of the working model. Comment and suggestion are collected from the customer and provided to the developer. Step 5: Refining prototype. • If the user is not happy with the current prototype, you need to refine the prototype according to the user's feedback and suggestions.This phase will not over until all therequirements specified by the user are met. Once the user is satisfied with the developed prototype, a final system is developed based on the approved final prototype. Step 6: Implement Product and Maintain. • Once the final system is developed based on the final prototype, it is thoroughly tested and deployed to production. The system undergoes routine maintenance for minimizing downtime and prevent large-scale failures. **Q17. What are the advantages and disadvantages of evolutionary prototyping model?** Advantages:• Prototype model need not know the detailed input, output, processes, adaptability of operating system and full machine interaction. • In the development process of this model users are actively involved. • The development process is the best platform to understand the system by the user. • Errors are detected much earlier. • Gives quick user feedback for better solutions. • It identifies the missing functionality easily. It also identifies the confusing or difficult functions. Disadvantages:• The client involvement is more and it is not always considered by the developer. • It is a slow process because it takes more time for development. • Many changes can disturb the rhythm of the development team. • It is a thrown away prototype when the users are confused with it. **Q18. What you mean by specialized process model?** Special process models take on many of the characteristics of one or more of the conventional models. However, specialized models tend to be applied when a narrowly defined software engineering approach is chosen. • Component based development • Formal methods model **Q19. What is Agile model in software engineering?** The Agile model was primarily designed to help a project to adapt to change requests quickly. So, the main aim of the Agile model is to facilitate quick project completion. To accomplish this task agility is required. Agility is achieved by fitting the process to the project, removing activities that may not be essential for a specific project. Also, anything that is wastage of time and effort is avoided. An agile process is adaptable and it should adapt incrementally S/w increments must be delivered in short time periods Agile principles, a. satisfy customer

b. welcome changing requirements, c. deliver working software frequently **Q20. what are the priciples of Agile model development in software engineering?** • Highest priority is to satisfy the customer through early and continuous delivery of valuable software. • It welcomes changing requirements, even late in development. • Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shortest timescale.• Build projects around motivated individuals. Give them the environment and the support they need, and trust them to get the job done. • Working software is the primary measure of progress. • Simplicity the art of maximizing the amount of work not done is essential. • The most efficient and effective method of conveying information to and within a development team is face-to-face conversation. **Q21. What are the different types of Agile development models?** • Extreme Programming(XP) • Adaptive Software Development • Scrum • Dynamic Systems Development Method • Feature Driven Development • Lean Software Development • Agile Modelling • Agile Unified Process **Q22. Explain about Extreme programming(XP) in Agile model?** It is the most widely used approach of agile development The XP Process includes • Planning • Design

• Coding • Testing

► XP design follows KIS(Keep It Simple) principle ► Encourages refactoring

► Pair programming concept is used in coding • Applications of Extreme Programming (XP): Some of the projects that are suitable to develop using XP model are given below: • Small projects: XP model is very useful in small projects consisting of small teams as face to face meeting is easier to achieve. • Projects involving new technology or Research projects: This type of projects face changing of requirements rapidly and technical problems. So XP model is used to complete this type of projects. **Q23. Explain ASD in agile model development?** Adaptive Software Development is cyclical like the Evolutionary model, with the phase names reflecting the unpredictability in the complex systems. Phases of ASD Life Cycle • Speculate • Collaborate • LearnThese three phases reflect the dynamic nature of Adaptive Software Development. The Adaptive Development explicitly replaces Determinism with Emergence. It goes beyond a mere change in lifecycle to a deeper change in management style. Adaptive Software Development has a dynamic Speculate-Collaborate-Learn Lifecycle. The Adaptive Software Development Lifecycle focuses on results, not tasks, and the results are identified as application features. ASD life cyle is shown below,Speculate:• In Adaptive Software Development, the term plan is replaced by the term speculate. While speculating, the team does not abandon planning, but it acknowledges the reality of uncertainty in complex problems. Speculate encourages exploration and experimentation. Iterations with short cycles are encouraged. Collaborate:• Collaborate would require the ability to work jointly to produce results, share knowledge or make decisions. Learn: • Team has to enhance their knowledge constantly, using practices such as – • Technical Reviews • Project Retrospectives • Customer Focus Groups **Q24. What is Scrum framework in agile model?** Scrum is a lightweight, iterative and incremental framework for managing complex work. Scrum framework activities: • Requirements • Analysis • Design • Evolution • Delivery Important concepts in scrum:, • Backlog: a prioritized list of requirements or features.Within each framework activity work task occurs within a process pattern called sprint. • Sprint: consists of work units required to achieve a requirement defined in backlog that can be done in a predefined time box(30days) • Scrum meeting : short meeting(15 minutes) held daily by the scrum Team • Demos: delivers the increment to the customer **Q25. Explain DSDM in Agile model?** • The Dynamic Systems Development method (DSDM)is an agile software development methodology. It is an iterative, incremental approach that is largely based on the Rapid Application Development (RAD) methodology. • Principles: . Focus on the business need . Deliver on time . Collaborate . Never compromise quality . Build incrementally from firm foundations . Develop iteratively . Communicate continuously and clearly . Demonstrate control. **Q26. Explain FDD in Agile model?** Feature Driven Development (FDD) is an agile framework that, as its name suggests, organizes software development around making progress on features. Features in the FDD context, though, are not necessarily product features in the commonly understood sense. "complete the login process" might be considered a feature in the Feature Driven Development (FDD) methodology. FDD was designed to follow a five-step development process, built largely around discrete "feature" projects. That project lifecycle looks like this: • Develop an overall model • Build a features list • Plan by feature • Design by feature • Build by feature **Q27. What is lean software development(LSD)?** Lean Software Development (LSD) is an agile framework based on optimizing development time and resources, eliminating waste, and ultimately delivering only what the product needs. The Lean approach is also often referred to as the Minimum Viable Product (MVP) strategy, in which a team releases a bare-minimum version of its product to the market, learns from users what they like, don't like and want to be added, and then iterates based on this feedback. **Q28. What is Agile modelling and AUP?** Agile modeling (AM) is a methodology for modeling and documenting software systems based on best practices. It is a collection of values and principles, that can be applied on an (agile) software development project. This methodology is more flexible than traditional modeling methods, making it a better fit in a fast changing environment.Agile Unified Process (AUP) is a simplified version of the Rational Unified Process (RUP), which is an iterative software development process framework. It describes a simple, easy to understand approach to developing business application software using agile techniques and **concepts.Q2. Draw Use Case Dingram of Result Management System of M.Tech. Programme.** Ans: **Q3. What are the components of state transition diagram ? Give example.** Ans: Figure – a state diagram for user verification. The basic purpose of a state diagram is to portray various changes in state of the class and not the processes or commands causing the changes. Basic components of a statechart diagram – Initial state – We use a black filled circle represent the initial state of a System or a class. Figure – initial state notation Transition – We use a solid arrow to represent the transition or change of control from one state to another. The arrow is labelled with the event which causes the change in state. Figure – transition. State – We use a rounded rectangle to represent a state. A state represents the conditions or circumstances of an object of a class at an instant of time. Figure – state notation Fork – We use a rounded solid rectangular bar to represent a Fork notation with incoming arrow from the parent state and outgoing arrows towards the newly created states. We use the fork notation to represent a state splitting into two or more concurrent states. Figure – a diagram using the fork notation. Join – We use a rounded solid rectangular bar to represent a Join notation with incoming arrows from the joining states and outgoing arrow towards the common goal state. We use the join notation when two or more states concurrently converge into one on the occurrence of an event or events.Figure – a diagram using join notation. Self transition – We use a solid arrow pointing back to the state itself to represent a self transition. There might be scenarios when the state of the object does not change upon the occurrence of an event. We use self transitions to represent such cases. Figure – self transition notation. Composite state – We use a rounded rectangle to represent a composite state also.We represent a state with internal activities using a composite state. Figure – a state with internal activities. Final state – We use a filled circle within a circle notation to represent the final state in a state machine diagram. Figure – final state notation **Q4. What are the different types of requirements ? Explain in detail.** Ans:A software requirement can be of 3 types: ● Functional requirements ● Non-functional requirements ● Domain requirements, /Functional Requirements: These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements. For example, in a hospital management system, a doctor should be able to retrieve the information of his patients. Each high-level functional requirement may involve several interactions or dialogues between the system and the outside world. In order to accurately describe the functional requirements, all scenarios must be enumerated. Non-functional requirements: These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements. They basically deal with issues like: ● Portability ● Security ● Maintainability ● Reliability ● Scalability ● Performance ● Reusability ● Flexibility. The process of specifying non-functional requirements requires the knowledge of the functionality of the system, as well as the knowledge of the context within which the system will operate. Domain requirements: Domain requirements are the requirements which are characteristic of a particular category or domain of projects. The basic functions that a system of a specific domain must necessarily exhibit come under this category. For instance, in an academic software that maintains records of a school or college, the functionality of being able to access the list of faculty and list of students of each grade is a domain requirement. These requirements are therefore identified from that domain model and are not user **Q6. What are the Objectives of Requirement Analysis ?** Ans: Requirement Analysis, also known as Requirement Engineering, is the process of defining user expectations for a new software being built or modified. In software engineering, it is sometimes referred to loosely by names such as requirements gathering or requirements capturing. Objectives of Requirement Analysis: .. Identify customer's needs. • Evaluate system for feasibility. • Perform economic and technical analysis. • Allocate functions to system elements. • Establish schedule and constraints. • Create system definitions. The various steps of requirement analysis are, (i) Draw the context diagram: The context diagram is a simple model that defines the boundaries and interfaces of the proposed systems with the external world. It identifies the entities outside the proposed system that interact with the system.   (ii) Development of a Prototype (optional): One effective way to find out what the customer wants is to construct a prototype, something that looks and preferably acts as part of the system they say they want. We can use their feedback to modify the prototype until the customer is satisfied continuously. Hence, the prototype helps the client to visualize the proposed system and increase the understanding of the requirements. When developers and users are not sure about some of the elements, a prototype may help both the parties to take a final decision. Some projects are developed for the general market. In such cases, the prototype should be shown to some representative sample of the population of potential purchasers. Even though a person who tries out a prototype may not buy the final system, but their feedback may allow us to make the product more attractive to others. The prototype should be built quickly and at a relatively low cost. Hence it will always have limitations and would not be acceptable in the final system. This is an optional activity. (iii) Model the requirements: This process usually consists of various graphical representations of the functions, data entities, external entities, and the relationships betweenthem. The graphical view may help to find incorrect, inconsistent, missing, and superfluous requirements. Such models include the Data Flow diagram, Entity-Relationship diagram, Data Dictionaries, State-transition diagrams, etc. (iv) Finalise the requirements: After modeling the requirements, we will have a better understanding of the system behavior. The inconsistencies and ambiguities have been identified and corrected. The flow of data amongst various modules has been analyzed. Elicitation and analyze activities have provided better insight into the system. Now we finalize the analyzed requirements, and the next step is to document these requirements in a prescribed format. **Q7. What kinds of errors are sought out during requirements validation? Explain.** Ans: • May be requirements are not consistent with the overall objectives for the System/product • Requirements may not be specified at the proper level of abstraction. That is, do some requirements provide a level of technical detail that is inappropriate at this stage? • Requirements that are unnecessary or does it represent an add-on feature that may not be essential to the objective of the system. • Unbounded and ambiguous requirements. • Requirements may conflict with other requirements. • There may be unachievable requirements. • Requirements that are not testable are considered as errors in validation. **Q8. Explain seven distinct requirements engineering functions.** Ans: The broad spectrum of tasks and techniques that lead to an understanding of requirements is called requirements engineering. It encompasses seven distinct tasks: inception, elicitation, elaboration, negotiation, specification, validation, and management. It is important to note that some of these tasks occur in parallel and all are adapted to the needs of the project. 1. Inception: ● Inception is a task where the requirement engineering asks a set of questions to establish a software process. ● In this task, it understands the problem and evaluates with the proper solution. ● It collaborates with the relationship between the customer and the developer. ● The developer and customer decide the overall scope and the nature of the question. 2. Elicitation:. Elicitation means to find the requirements from anybody .The requirements are difficult because the following problems occur in elicitation. Problem of scope: The customer give the unnecessary technical detail rather than clarity of the overall system objective.Problem of understanding: Poor understanding between the customer and the developer regarding various aspect of the project like capability, limitation of the computing environment. Problem of volatility: In this problem, the requirements change from time to time and it is difficult while developing the project. 3. Elaboration:● In this task, the information taken from user during inception and elaboration and are expanded and refined in elaboration. ● Its main task is developing pure model of software using functions, feature and constraints of a software. 4. Negotiation :● In negotiation task, a software engineer decides the how will the project be achieved with limited business resources. ● To create rough guesses of development and access the impact of the requirement on the project cost and delivery time. 5. Specification: ● In this task, the requirement engineer constructs a final work product. ● The work product is in the form of software requirement specification. ● In this task, formalize the requirement of the proposed software such as informative, functional and behavioral. ● The requirements are formalize in both graphical and textual formats. 6. Validation ● The work product is built as an output of the requirement engineering and that is accessed for the quality through a validation step. ● The formal technical reviews from the software engineer, customer and other stakeholders helps for the primary requirements validation mechanism. 7. Requirement management:● It is a set of activities that help the project team to identify, control and track the requirements and changes can be made to the requirements at any time of the ongoing project. ● These tasks start with the identification and assign a unique identifier to each of the requirement. ● After finalizing the requirement traceability table is developed. ● The examples of traceability table are the features, sources, dependencies, subsystems and interface of the requirement.**Q9. How will you negotiate the requirements?** Ans: In an ideal requirements engineering context, the inception, elicitation, and elaboration tasks determine customer requirements in sufficient detail to proceed to subsequent Software engineering activities. Unfortunately, this rarely happens. In reality, You may have to enter into a negotiation with one or more stakeholders. In most cases, stakeholders are asked to balance functionality, performance, and other product or system characteristics against cost and time-to-market. The intent of this negotiation is to develop a project plan that meets stakeholder needs while at the same time reflecting the real-world constraints (e.g., time, people, budget) that have been placed on the software team. The best negotiations strive for a “win-win” result. That is, stakeholders win by getting the system or product that satisfies the majority of their needs and you (as a member of the software team) win by working to realistic and achievable budgets and deadlines. There are a set of negotiation activities at the beginning of each software process iteration. Rather than a single customer communication activity, the following activities are defined: 1. Identification of the system or subsystem’s key stakeholders. 2. Determination of the stakeholders’ “win conditions.” 3. Negotiation of the stakeholders’ win conditions to reconcile them into a set of win-win conditions for all concerned (including the software team). Successful completion of these initial steps achieves a win-win result, which becomes the key criterion for proceeding to subsequent software engineering activities. **Q10. What do you mean by Quality Function Deployment (QFD) ?** Ans: Quality function deployment (QFD) is a quality management technique that translates the needs of the customer into technical requirements for software. QFD “concentrates on maximizing customer satisfaction from the software engineering process”. QFD identifies three types of requirements: Normal requirements. The objectives and goals that are stated for a product or system during meetings with the customer. If these requirements are present, the customer is satisfied. Examples of normal requirements might be requested types of graphical displays, specific system functions, and defined levels of performance. Expected requirements. These requirements are implicit to the product or system and may be so fundamental that the customer does not explicitly state them. Their absence will be a cause for significant dissatisfaction. Examples of expected requirements are: ease of human/machine interaction, overall operational correctness and reliability, and ease of softwareinstallation. Exciting requirements.These features go beyond the customer’s expectations and prove to be very satisfying when present. For example, software for a new mobile phone comes with standard features, but is coupled with a set of unexpected capabilities (e.g., multitouch screen, visual voice mail) that delight every user of the product.QFD uses customer interviews and observation, surveys, and examination of historical data (e.g., problem reports) as raw data for the requirements gathering activity. These data are then translated into a table of requirements—called the customer voice table—that is reviewed with the customer and other stakeholders. **Q11. How the requirements model is built?** Ans: The intent of the analysis model is to provide a description of the required informational, functional, and behavioral domains for a computer-based system. The model changes dynamically as you learn more about the system to be built, and other stakeholders understand more about what they really require. As the requirements model evolves, certain elements will become relatively stable, providing a solid foundation for the design tasks that follow. Elements of the Requirements

Model :● Scenario-based elements. The system is described from the user's point of view using a scenario-based approach. For example, basic use cases and their corresponding use-case diagrams evolve into more elaborate template-based use cases. Scenario-based elements of the requirements model are often the first part of the model that is developed. As such, they serve as input for the creation of other modeling elements. ● Class-based elements. Each usage scenario implies a set of objects that are manipulated as an actor interacts with the system. These objects are categorized into classes—a collection of things that have similar attributes andcommonbehaviors ● Behavioral elements. The behavior of a computer-based system can have a profound effect on the design that is chosen and the implementation approach that is applied. Therefore, the requirements model must provide modeling elements that depictbehavior.The state diagram is one method for representing the behavior of a system by depicting its states and the events that cause the system to change state. A state is any externally observable mode of behavior. ● Flow-oriented elements. Information is transformed as it flows through a computer-based system. The system accepts input in a variety of forms, applies functions to transform it, and produces output in a variety of forms. Input may be a control signal transmitted by a transducer, a series of numbers typed by a human operator, a packet of information transmitted on a network link, or a voluminous data file retrieved from secondary storage. The transform(s) may comprise a single logical comparison, a complex numerical algorithm, or a rule-inference approach of an expert system. Output may light a single LED or produce a 200-page report. **Q12. How will you elicit the requirments. Discuss** Ans: Requirements elicitation (also called requirements gathering) combines elements of problem solving, elaboration, negotiation, and specification. In order to encouragea collaborative, team-oriented approach to requirements gathering, stakeholders work together to identify the problem, propose elements of the solution, negotiate different approaches and specify a preliminary set of solution requirements. There are various ways to discover requirements. Interviews :Interviews are strong medium to collect requirements. Organization may conduct several types of interviews such as: ● Structured (closed) interviews, where every single information to gather is decided in advance, they follow pattern and matter of discussion firmly. ● Non-structured (open) interviews, where information to gather is not decided in advance, more flexible and less biased. ● Oral interviews ● Written interviews ● One-to-one interviews which are held between two persons across the table. ● Group interviews which are held between groups of participants. They help to uncover any missing requirement as numerous people are involved. Surveys: Organization may conduct surveys among various stakeholders by querying about their expectation and requirements from the upcoming system. Questionnaires:A document with pre-defined set of objective questions and respective options is handed over to all stakeholders to answer, which are collected and compiled. A shortcoming of this technique is, if an option for some issue is not mentioned in the questionnaire, the issue might be left unattended. Task analysis: Team of engineers and developers may analyze the operation for which the new system is required. If the client already has some software to perform certain operation, it is studied and requirements of proposed system are collected. Domain Analysis: Every software falls into some domain category. The expert people in the domain can be a great help to analyze general and specific requirements. Brainstorming :An informal debate is held among various stakeholders and all their inputs are recorded for further requirements analysis.Prototyping: Prototyping is building user interface without adding detail functionality for user to interpret the features of intended software product. It helps giving better idea of requirements. If there is no software installed at client's end for developer's reference and the client is not aware of its own requirements, the developer creates a prototype based on initially mentioned requirements. The prototype is shown to the client and the feedback is noted. The client feedback serves as an input for requirement gathering. Observation :Team of experts visit the client's organization or workplace. They observe the actual working of the existing installed systems. They observe the workflow at client's end and how execution problems are dealt. The team itself draws some conclusions which aid to form requirements expected from the software. **Q15. Explain requirement gathering techniques.** Ans: There are a number of requirements elicitation methods. Few of them are listed below – 1. Interviews 2. Brainstorming Sessions3. Facilitated Application Specification Technique (FAST) 4. Quality Function Deployment (QFD) 5. Use Case Approach. The success of an elicitation technique used depends on the maturity of the analyst, developers, users, and the customer involved. 1.Interviews: Objective of conducting an interview is to understand the customer's expectations from the software. It is impossible to interview every stakeholder hence representatives from groups are selected based on their expertise and credibility. Interviews maybe be open ended or structured. 1. In open ended interviews there is no pre-set agenda. Context free questions may be asked to understand the problem. 2. In structured interview, agenda of fairly open questions is prepared. Sometimes a proper questionnaire is designed for the interview. 2. Brainstorming Sessions: • It is a group technique • It is intended to generate lots of new ideas hence providing a platform to share views • A highly trained facilitator is required to handle group bias and group conflicts. • Every idea is documented so that everyone can see it. • Finally a document is prepared which consists of the list of requirements and their priority if possible.3.FacilitatedApplicationSpecificationTechnique: It's objective is to bridge the expectation gap – difference between what the developers think they are supposed to build and what customers think they are going to get. A team oriented approach is developed for requirements gathering. Each attendee is asked to make a list of objects that are- 1. Part of the environment that surrounds the system 2. Produced by the system 3. Used by the system Each participant prepares his/her list, different lists are then combined, redundant entries are eliminated, team is divided into smaller sub-teams to develop mini-specifications and finally a draft of specifications is written down using all the inputs from themeeting.4.QualityFunctionDeployment: In this technique customer satisfaction is of prime concern, hence it emphasizes on therequirements which are valuable to the customer. 3 types of requirements are identified – • Normal requirements – In this the objective and goals of the proposed software are discussed with the customer. Example – normal requirements for a result management system may be entry of marks, calculation of results, etc • Expected requirements – These requirements are so obvious that the customer need not explicitly state them. Example – protection from unauthorized access. • Exciting requirements – It includes features that are beyond customer's expectations and prove to be very satisfying when present. Example – when unauthorized access is detected, it should backup and shutdown all processes. The major steps involved in this procedure are –   1. Identify all the stakeholders, eg. Users, developers, customers etc 2. List out all requirements from customer. 3. A value indicating degree of importance is assigned to each requirement. 4. In the end the final list of requirements is categorized as – • It is possible to achieve • It should be deferred and the reason for it • It is impossible to achieve and should be dropped off 5.UseCaseApproach: This technique combines text and pictures to provide a better understanding of the requirements. The use cases describe the 'what', of a system and not 'how'. Hence they only give a functional view of the system. The components of the use case design includes three major things – Actor, Use cases, use case diagram. 1. Actor – It is the external agent that lies outside the system but interacts with it in some way. An actor maybe a person, machine etc. It is represented as a stick figure. Actors can be primary actors or secondary actors. • Primary actors – It requires assistance from the system to achieve a goal. • Secondary actor – It is an actor from which the system needs assistance. 2. Use cases – They describe the sequence of interactions between actors and the system. They capture who(actors) do what(interaction) with the system. A complete set of use cases specifies all possible ways to use the system. 3. Use case diagram – A use case diagram graphically represents what happens when an actor interacts with a system. It captures the functional aspect of the system. • A stick figure is used to represent an actor.• An oval is used to represent a use case. • A line is used to represent a relationship between an actor and a use case. **Q18. What is SRS? Explain the need for SRS.** Ans: The production of the requirements stage of the software development process isSoftware Requirements Specifications (SRS) (also called a requirements document). This report lays a foundation for software engineering activities and is constructing when entire requirements are elicited and analyzed. SRS is a formal report, which acts as a representation of software that enables the customers to review whether it (SRS) is according to their requirements. Also, it comprises user requirements for a system as well as detailed specifications of the system requirements. SRS serves as an input our other all documents created in later stages of software development life cycle. ● It is a feedback to the customer. ● It's modularized the problem statement. ● It the bases of system design. ● It defines product scope. **Q19. What is the importance of requirement analysis ? What are the problems in requirement that the analyst needs to identify ?** Ans: Requirements analysis results in the specification of software's operational characteristics,indicates software's interface with other system elements, and establishes constraints that software must meet. Requirements analysis allows you (regardless of whether you're called a software engineer, an analyst, or a modeler) to elaborate on basic requirements established during the inception, elicitation, and negotiation tasks that are part of requirements engineering.Requirements analysis can be a long and tiring process during which many delicate psychological skills are involved. New systems change the environment and relationships between people, so it is important to identify all the stakeholders, take into account all their needs and ensure they understand the implications of the new systems. Analysts can employ several techniques to elicit the requirements from the customer. These may include the development of scenarios (represented as user stories in agile methods), the identification of use cases, the use of workplace observation or ethnography, holding interviews, or focus groups (more aptly named in this context as requirements workshops, or requirements review sessions) and creating requirements lists. Prototyping may be used to develop an example system that can be demonstrated to stakeholders. Where necessary, the analyst will employ a combination of these methods to establish the exact requirements of the stakeholders, so that a system that meets the business needs is produced. Problem of scope: The customer give the unnecessary technical detail rather than clarity of the overall system objective.Problem of understanding: Poor understanding between the customer and the developer regarding various aspect of the project like capability, limitation of the computing environment. Problem of volatility: In this problem, the requirements change from time to time and it is difficult while developing the project.Modeling with UML: UML is popular for its diagrammatic notations. We all know that UML is for visualizing, specifying, constructing and documenting the components of software and non-software systems. Hence, visualization is the most important part which needs to be understood and remembered. UML notations are the most important elements in modeling. Efficient and appropriate use of notations is very important for making a complete and meaningful model. The model is useless, unless its purpose is depicted properly. Hence, learning notations should be emphasized from the very beginning. Different notations are available for things and relationships. UML diagrams are made using the notations of things and relationships. Extensibility is another important feature which makes UML more powerful and flexible. It is very important to distinguish between the UML model. Different diagrams are used for different types of UML modeling. There are three important types of UML modeling. Structural Modeling :Structural modeling captures the static features of a system. They consist of the following – Classes diagrams •Deployment diagrams •Package diagrams •Component diagram. Structural model represents the framework for the system and this framework is the place where all other components exist. Hence, the class diagram, component diagram and deployment diagrams are part of structural modeling. They all represent the elements and the mechanism to assemble them. The structural model never describes the dynamic behavior of the system. Class diagram is the most widely used structural diagram. Behavioural Modeling

:Behavioral model describes the interaction in the system. It represents the interaction among the structural diagrams. Behavioral modeling shows the dynamic nature of the system. They consist of the following − Activity diagrams •Interaction diagrams •Use case diagrams •All the above show the dynamic sequence of flow in a system. Architectural Modeling :Architectural model represents the overall framework of the system. It contains both structural and behavioral elements of the system. Architectural model can be defined as the blueprint of the entire system. Package diagram comes under architectural modeling.

Behavioural modeling:1. Usecase diagram A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different usecases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses.Also the user is actor in usecase.Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. Hence, when a system is analyzed to gather its functionalities, use cases are prepared and actors are identified. When the initial task is complete, use case diagrams are modelled to present the outside view. How to Draw a Use Case Diagram? Use case diagrams are considered for high level requirement analysis of a system. When the requirements of a system are analyzed, the functionalities are captured in use cases. We can say that use cases are nothing but the system functionalities written in an organized manner. The second thing which is relevant to use cases are the actors. Actors can be defined as something that interacts with the system. Actors can be a human user, some internal applications, or may be some external applications. When we are planning to draw a use case diagram, we should have the following items identified. • Functionalities to be represented as use case • Actors • Relationships among the use cases and actors. Picture1, 2. Interaction digram. From the term Interaction, it is clear that the diagram is used to describe some type of interactions among the different elements in the model. This interaction is a part of dynamic behavior of the system. This interactive behavior is represented in UML by two diagrams known as Sequence diagram and Collaboration diagram. The basic purpose of both the diagrams are similar. Sequence diagram emphasizes on time sequence of messages and collaboration diagram emphasizes on the structural organization of the objects that send and receive messages. The purpose of interaction diagram is − • To capture the dynamic behaviour of a system. • To describe the message flow in the system. • To describe the structural organization of the objects. • To describe the interaction among objects.3.Activity digram: Activity diagram is another important diagram in UML to describe the dynamic aspects of the system. Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc.The basic purposes of activity diagrams is similar to other two diagrams. It captures the dynamic behavior of the system. Other four diagrams are used to show the message flow from one object to another but activity diagram is used to show message flow from one activity to another. Activity is a particular operation of the system. Activity diagrams are not only used for visualizing the dynamic nature of a system, but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in the activity diagram is the message part.Software design :Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation. For assessing user requirements, an SRS (Software Requirement Specification) document is created whereas for coding and implementation, there is a need of more specific and detailed requirements in software terms. The output of this process can directly be used into implementation in programming languages. Software design is the first step in SDLC (Software Design Life Cycle), which moves the concentration from problem domain to solution domain. It tries to specify how to fulfill the requirements mentioned in SRS. objectives of Software Design:1. Correctness:A good design should be correct i.e. it should correctly implement all the functionalities of the system.2. Efficiency:A good software design should address the resources, time and cost optimization issues.3. Understandability:A good design should be easily understandable for which it should be modular and all the modules are arranged in layers.4. Completeness:The design should have all the components like data structures, modules, and external interfaces, etc.5. Maintainability:A good software design should be easily amenable to change whenever a change request is made from the customer side.Software Design Concepts:The software design concept simply means the idea or principle behind the design. It describes how you plan to solve the problem of designing software, the logic, or thinking behind how youwill design software. It allows the software engineer to create the model of the system or software or product that is to be developed or built.1. Abstraction- hide relevant data. Abstraction simply means to hide the details to reduce complexity and increases efficiency or quality. Different levels of Abstraction are necessary and must be applied at each stage of the design process so that any error that is present can be removed to increase the efficiency of the software solution and to refine the software solution2.Modularity- subdivide the system . Modularity simply means to divide the system or project into smaller parts to reduce the complexity of the system or project. In the same way, modularity in design means to subdivide a system into smaller parts so that these parts can be created independently and then use these parts in different systems to perform different functions. 3.Architecture- design a structure of something. Architecture simply means a technique to design a structure of something. Architecture in designing software is a concept that focuses on various elements and the data of the structure. These components interact with each other and use the data of the structure in architecture.4.Refinement- removes impurities. Refinement simply means to refine something to remove any impurities if present and increase the quality. The refinement concept of software design is actually a process of developing or presenting the software or system in a detailed manner that means to elaborate a system or software. Refinement is very necessary to find out any error if present and then to reduce it.5.Information Hiding- hide the information . Information hiding simply means to hide the information so that it cannot be accessed by an unwanted party. In software design, information hiding is achieved by designing the modules in a

manner that the information gathered or contained in one module is hidden and it cant be accessed by any other modules.5.Pattern- a repeated form. The pattern simply means a repeated form or design in which the same shape is repeated several times to form a pattern. The pattern in the design process means the repetition of a solution to a common recurring problem within a certain context.6.Refactoring- reconstruct something. Refactoring simply means to reconstruct something in such a way that it does not affect the behavior or any other features. Refactoring in software design means to reconstruct the design to reduce and complexity and simplify it without affecting the behavior or its functions. Fowler has defined refactoring as “the process of changing a software system in a way that it won’t affect the behavior of the design and improves the internal structure”.design models/elements 1. Data design elements • The data design element produced a model of data that represent a high level of abstraction. • This model is then more refined into more implementation specific representation which is processed by the computer based system. • The structure of data is the most important part of the software design. 2. Architectural design elements:• The architecture design elements provides us overall view of the system. • The architectural design element is generally represented as a set of interconnected subsystem that are derived from analysis packages in the requirement model. The architecture model is derived from following sources: • The information about the application domain to built the software. • Requirement model elements like data flow diagram or analysis classes, relationship and collaboration between them. • The architectural style and pattern as per availability. 3. Interface design elements • The interface design elements for software represents the information flow within it and out of the system. • They communicate between the components defined as part of architecture. Following are the important elements of the interface design: 1. The user interface 2. The external interface to the other systems, networks etc. 3. The internal interface between various components. 4. Component level diagram elements • The component level design for software is similar to the set of detailed specification of each room in a house.• The component level design for the software completely describes the internal details of the each software component. • The processing of data structure occurs in a component and an interface which allows all the component operations. • In a context of object-oriented software engineering, a component shown in a UML diagram. • The UML diagram is used to represent the processing logic. 5. Deployment level design elements • The deployment level design element shows the software functionality and subsystem that allocated in the physical computing environment which support the software. • Following figure shows three computing environment as shown. These are the personal computer, the CPI server and the Control panel.**Coupling and Cohesion**: When a software program is modularized, its tasks are divided into several modules based on some characteristics. As we know, modules are set of instructions put together in order to achieve some tasks. They are though, considered as single entity but may refer to each other to work together. There are measures by which the quality of a design of modules and their interaction among them can be measured. These measures are called coupling and cohesion. **Cohesion**: Cohesion is a measure that defines the degree of intra-dependability within elements of a module. The greater the cohesion, the better is the program design. There are seven types of cohesion, namely – • Co-incidental cohesion - It is unplanned and random cohesion, which might be the result of breaking the program into smaller modules for the sake of modularization. Because it is unplanned, it may serve confusion to the programmers and is generally not-accepted. • Logical cohesion - When logically categorized elements are put together into a module, it is called logical cohesion. • Temporal Cohesion - When elements of module are organized such that they are processed at a similar point in Time, it is called temporal cohesion. • Procedural cohesion - When elements of module are grouped together, which are executed sequentially in order to perform a task, it is called procedural cohesion. • Communicational cohesion - When elements of module are grouped together, which are executed sequentially and work on same data (information), it is called communicational cohesion. • Sequential cohesion - When elements of module are grouped because the output of one element serves as input to another and so on, it is called sequential cohesion. • Functional cohesion - It is considered to be the highest degree of cohesion, and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well-defined function. It can also be reused .**Coupling** :Coupling is a measure that defines the level of inter-dependability among modules of a program. It tells at what level the modules interfere and interact with each other. The lower the coupling, the better the program. There are five levels of coupling, namely - • Content coupling - When a module can directly access or modify or refer to the content of another module, it is called content level coupling. • Common coupling- When multiple modules have read and write access to some global data, it is called common or global coupling. • Control coupling- Two modules are called control-coupled if one of them decides the function of the other module or changes its flow of execution. • Stamp coupling- When multiple modules share common data structure and work on different part of it, it is called stamp coupling. • Data coupling- Data coupling is when two modules interact with each other by means of passing data (as parameter). If a module passes data structure as parameter, then the receiving module should use all its components. Ideally, no coupling is considered to be the best.**Q2. Explain coupling and cohesion** Ans:**Coupling**: Coupling is the measure of the degree of interdependence between the modules. A good software will have low coupling. Types of Coupling: ● Data Coupling: If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled. In data coupling, the components are independent to each other and communicating through data. Module communications don't contain tramp data. Example-customer billing system. ● Stamp Coupling :In stamp coupling, the complete data structure is passed from one module to another module. Therefore, it involves tramp data. It may be necessary due to efficiency factors- this choice made by the insightful designer, not a lazy programmer. ● Control Coupling: If the modules communicate by passing control information, then they are said to be control coupled. It can be bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality. Example- sort function that takes comparison function as an argument. ● External Coupling: In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware. Ex- protocol, external file, device format, etc.● Common Coupling: The modules have shared data such as global data structures.The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change. So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses and reduced maintainability. ● Content Coupling: In a content coupling, one module can modify the data of another module or control flow is passed from one module to the other module. This is the worst form of coupling and should be avoided. **Cohesion**: Cohesion is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Basically, cohesion is the internal glue that keeps the module together. A good software design will have high cohesion. Types of Cohesion: ● Functional Cohesion: Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. It is an ideal situation. ● Sequential Cohesion: An element outputs some data that becomes the input for other element, i.e., data flow between the parts. It occurs naturally in functional programming languages. ● Communicational Cohesion: Two elements operate on the same input data or contribute towards the same output data. Example- update record int the database and send it to the printer. ● Procedural Cohesion: Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable. Ex- calculate student GPA, print student record, calculate cumulative GPA, print cumulative GPA.● Temporal Cohesion: The elements are related by their timing involved. A module connected with temporal cohesion all the tasks must be executed in the same time-span. This cohesion contains the code for initializing all the parts of the system. Lots of different activities occur, all at init time. ● Logical Cohesion: The elements are logically related and not functionally. Ex- A component reads inputs from tape, disk, and network. All the code for these functions is in the same component. Operations are related, but the functions are significantly different. 3. Coincidental Cohesion: The elements are not related(unrelated). The elements have no conceptual relationship other than location in source code. It is accidental and the worst form of cohesion. Ex- print next line and reverse the characters of a string in a single component. **Q4. Define Coupling**. Ans: . Coupling is the measure of the degree of interdependence between the modules. A good software will have low coupling. 5. How software cost is planned based on cost estimation techniques ? Ans: The objective of software project planning is to provide a framework that enables themanager to make reasonable estimates of resources, cost, and schedule. In addition, estimates should attempt to define best-case and worst-case scenarios so that project outcomes can be bounded. Although there is an inherent degree of uncertainty, the software team embarks on a plan that has been established as a consequence of these tasks. Therefore, the plan must be adapted and updated as the project proceeds. Software cost and effort estimation will never be an exact science. Too many variables—human, technical, environmental, political—can affect the ultimate cost of software and effort applied to develop it. Cost estimation of software development project focuses on how associating estimates of effort and time with the project activities. Estimation involves answering the following questions [1]: • How much effort is required to complete each activity? • How much calendar time is needed to complete each activity? • What is the total cost of each activity? Project cost estimation and project scheduling are usually carried out together. The costs of development are primarily the costs of the effort involved, so the effort computation is used in both the cost and the schedule estimate. The initial cost estimates may be used toestablish a budget for the project and to set a price for the software for a customer. The total cost of a software development project is the sum of following costs: • Hardware and software costs including maintenance. • Travel and training costs. • Effort costs of paying software developers. For most projects, the dominant cost is the effort cost. Effort costs are not just the salaries of the software engineers who are involved in the project. The following overhead costs are all part of the total effort cost: • Costs of heating and lighting offices. • Costs of support staff (accountants, administrators, system managers, cleaners, technicians etc.). • Costs of networking and communications. • Costs of central facilities (library, recreational facilities, etc.). • Costs of social security and employee benefits such as pensions and health insurance. Cost estimation techniques: • Algorithmic cost modelling: Relating some software metric a mathematical model is developed to estimate the project cost. • Expert judgement: Several experts on the proposed software development techniques and the application domain are asked to estimate the project cost. The estimation process iterates until an agreed estimate is reached. • Estimation by previous projects. The cost of a new project is estimated by a completed project in the same application domain. • Application of Parkinson's Law :Parkinson's Law states that work expands to fill the time available and the cost is determined by the resources used.• Pricing to win:The software cost is estimated by the price what the customer has available to spend on the project. **Q6. What are the techniques that software engineering to tackle the problem of exponential growth of problem complexity with its size** Ans:Software engineering principles use two important techniques to reduce problem complexity: abstraction and decomposition. The principle of abstraction implies that a problem can be simplified by omitting irrelevant details. Once simpler problem is solved then the omitted details can be taken into consideration to solve the next lower level abstraction. In this technique any random decomposition of a problem into smaller parts will not help. The problem has to bedecomposed such that each component of the decomposed problem can be solved in solution and then the solution of the different components can be combined to get the full solution.In other words, a good decomposition as shown in should minimize interactions among various components. **Q7. What is the goal of concurrency control ?** Ans: Concurrency in software engineering means the collection of techniques and mechanisms that enable a computer program to perform several different tasks simultaneously, or apparently simultaneously. This allows for parallel execution of the concurrent units, which can significantly improve overall speed of the execution. The base goals of concurrent programming include correctness, performance and robustness. **Q8. Briefly explain the role of GOTO statements in structured coding ?** Ans: A goto statement provides an unconditional jump from the 'goto' to a labeled statement in the same function. NOTE – Use of goto statement is highly discouraged in any programming language because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a goto can be rewritten to avoid them. Syntax: The syntax for a goto statement is as follows – goto label;

..

.

label: statement;Here label can be any plain text except keyword and it can be set anywhere in the program above or below to goto statement. Flow Diagram **Q9. Briefly explain COCOMO model** Ans: Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e number of Lines of Code. It is a procedural cost estimate model for software projects and often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time and quality. These are types of COCOMO model: 1. Basic COCOMO Model 2. Intermediate COCOMO Model 3. Detailed COCOMO Model. The first level, Basic COCOMO can be used for quick and slightly rough calculations of Software Costs. Its accuracy is somewhat restricted due to the absence of sufficient factor considerations. Intermediate COCOMO takes these Cost Drivers into account and Detailed COCOMO additionally accounts for the influence of individual project phases, i.e in case of Detailed it accounts for both these cost drivers and also calculations are performed phase wise henceforth producing a more accurate result. These two models are further discussed below. Estimation of Effort: Calculations –Basic Model – The above formula is used for the cost estimation of for the basic COCOMO model, and also is used in the subsequent models. The constant values a,b,c and d for the Basic Model for the different categories of system: Software Projects a b c d Organic 2.4 1.05 2.5 0.38. Semi Detached 3.0 1.12 2.5 0.35. Embedded 3.6 1.20 2.5 0.32. The effort is measured in Person-Months and as evident from the formula is dependent on Kilo-Lines of code. The development time is measured in Months. These formulas are used as such in the Basic Model calculations, as not much consideration of different factors such as reliability, expertise is taken into account, henceforth the estimate is rough. **Intermediate Model –** The basic Cocomo model assumes that the effort is only a function of the number of lines of code and some constants evaluated according to the different software system. However, in reality, no system's effort and schedule can be solely calculated on the basis of Lines of Code. For that, various other factors such as reliability, experience, Capability. The Intermediate COCOMO formula now takes the form: The values of a and b in case of the intermediate model are as follows: Software Projects a b, /Organic 3.2 1.05 Semi Detached 3.0 1.12 Embeddedc 2.8 1.20 **Q10. What are organic, semidetached, and embedded systems:** Ans: Organic – A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem. Semi-detached – A software project is said to be a Semi-detached type if the vital characteristics such as team-size, experience, knowledge of the various programming environment lie in between that of organic and Embedded. The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience and better guidance and creativity. Eg: Compilers or different Embedded Systems can be considered of Semi-Detached type. Embedded – A software project with requiring the highest level of complexity, creativity, and experience requirement fall under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models. **Q11. What are user defined data types** Ans: User-Defined DataTypes: The data types that are defined by the user are called the derived datatype or user-defined derived data type. These types include: ● Class ● Structure ● Union ● Enumeration ● Typedef defined DataType. Below is the detailed description of the following types:, /Class: The building block that leads to Object-Oriented programming is a Class. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.Syntax: Structure: A structure is a user defined data type in C/C . A structure creates a data type that can be used to group items of possibly different types into a single type. Syntax: struct address {, /char name[50];, /char street[100]; char city[50];, /char state[20];, /int pin;, /}; Union: Like Structures, union is a user defined data type. In union, all members share the same memory location. For example in the following C program, both x and y share the same location. If we change x, we can see the changes being reflected in y. Enumeration: Enumeration (or enum) is a user defined data type in C. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain. Syntax: enum State {Working = 1, Failed = 0}; Typedef: C allows you to define explicitly new data type names by using the keyword typedef. Using typedef does not actually create a new data class, rather it defines a name for an existing type. This can increase the portability(the ability of a program to be used across different types of machines; i.e., mini, mainframe, micro, etc; without much changes into the code)of a program as only the typedef statements would have to be changed. Using typedef one can also aid in self-documenting code by allowing descriptive names for the standard data types. Syntax: Typedef type name; wheretype is any C data type and name is the new name for this data type. This defines another name for the standard type ofC .**Q12. What are the standards and guidelines for coding styles ?** Ans: Indentation: Proper and consistent indentation is essential in producing easy to read and maintainable programs. Indentation should be used to: ○ Emphasize the body of a control structure such as a loop or a select statement. ○ Emphasize the body of a conditional statement ○ Emphasize a new scope block. Inline comments: Inline comments analyze the functioning of the subroutine, or key aspects of the algorithm shall be frequently used. Rules for limiting the use of global: These rules file what types of data can be declared global and what cannot. Structured Programming:Structured (or Modular) Programming methods shall be used. "GOTO" statements shall not be used as they lead to "spaghetti" code, which is hard to read and maintain, except as outlined line in the FORTRAN Standards and Guidelines. Naming conventions for global variables, local variables, and constant identifiers: A possible naming convention can be that global variable names always begin with a capital letter, local variable names are made of small letters, and constant names are always capital letters. Error return conventions and exception handling system: Different functions in a program report the way error conditions are handled should be standard within an organization. For example, different tasks while encountering an error condition should either return a 0 or 1 consistently. **Coding Guidelines:**General coding guidelines provide the programmer with a set of the best methods which can be used to make programs more comfortable to read and maintain. Most of the examples use the C language syntax, but the guidelines can be tested to all languages. Line Length: It is considered a good practice to keep the length of source code lines at or below 80 characters. Lines longer than this may not be visible properly on some terminals and tools. Some printers will truncate lines longer than 80 columns. Spacing: The appropriate use of spaces within a line of code can improve readability. Example:

```
Bad:       cost=price (price*sales_tax)

fprintf(stdout ,"The total cost is %5.2f\n",cost);

Better:     cost = price ( price * sales_tax )

fprintf (stdout,"The total cost is %5.2f\n",cost);
```

The code should be well-documented: As a rule of thumb, there must be at least one comment line on the average for every three-source line. The length of any function should not exceed 10 source lines: A very lengthy function is generally very difficult to understand as it possibly carries out many various functions. For the same reason, lengthy functions are possible to have a disproportionately larger number of bugs. Do not use goto statements: Use of goto statements makes a program unstructured and very tough to understand. Inline Comments: Inline comments promote readability. Error Messages: Error handling is an essential aspect of computer programming. This does not only include adding the necessary logic to test for and handle errors but also involves making error messages meaningful. **Q14. State three important advantages of structured programming**. Ans: The following are the different advantages of structured programming 1. It is user friendly and easy to understand. 2. Similar to English vocabulary of words and symbols. 3. It is easier to learn. 4. They require less time to write. 5. They are easier to maintain. 6. These are mainly problem oriented rather than machine based. 7. Program written in a higher level language can be translated into many machine languages and therefore can run on any computer for which there exists an appropriate translator. 8. It is independent of machine on which it is used i.e. programs developed in high level languages can be run on any computer.**Q16. Explain any two empirical estimation techniques** Ans:Expert Judgment Technique: Expert judgment is one of the most widely used estimation techniques. In this approach, an expert makes an educated guess of the problem size after analyzing the problem thoroughly. Usually, the expert estimates the cost of the different components (i.e. modules or subsystems) of the system and then combines them to arrive at the overall estimate. However, this technique is subject to human errors and individual bias. Also, it is possible that the expert may overlook some factors inadvertently. Delphi cost estimation Delphi estimation iscarried out by a team comprising of a group of experts and a coordinator. In this approach, the coordinator provides each estimator with a copy of the software requirements specification (SRS) document and a form for recording his cost estimate. Estimators complete their individual estimates anonymously and submit to the coordinator. In their estimates, the estimators mention any unusual characteristic of the product which has influenced his estimation. The coordinator prepares and distributes the summary of the responses of all the estimators, and includes any unusual rationale noted by any of the estimators. Based on this summary, the estimators re-estimate. This process is iterated for several rounds. However, no discussion among the estimators is allowed during the entire estimation process. **Q17. What is project planning ? Explain various project estimation techniques**. Ans:Project planning: Once a project is found to be feasible, software project managers undertake project planning. Project planning is undertaken and completed even before any development activity starts. Project planning consists of the following essential activities: • Estimating the following attributes of the project: Project size: What will be problem complexity in terms of the effort and time required to develop the product? Cost: How much is it going to cost to develop the project? Duration: How long is it going to take to complete development? Effort: How much effort would be required?The effectiveness of the subsequent planning activities is based on the accuracy of these estimations. • Scheduling manpower and other resources • Staff organization and staffing plans • Risk identification, analysis, and abatement planning .. Miscellaneous plans such as quality assurance plan, configuration management plan, etc. Metrics for software project size estimation Accurate estimation of the problem size is fundamental to satisfactory estimation of effort, time duration and cost of a software project. In order to be able to accurately estimate the project size, some important metrics should be defined in terms of which the project size can be expressed. Currently two metrics are popularly being used widely to estimate size: lines of code (LOC) and function point (FP). Lines of Code (LOC)LOC is the simplest among all metrics available to estimate project size. This metric is very popular because it is the simplest to use. Using this metric, the project size is estimated by counting the number of source instructions in the developed program. Obviously, while counting the number of source instructions, lines used for commenting the code and the header lines should be ignored. Function point (FP)One of the important advantages of using the function point metric is that it can be used to easily estimate the size of a software product directly from the problem specification. The conceptual idea behind the function point metric is that the size of a software product is directly dependent on the number of different functions or features it supports. A software product supporting many features would certainly be of larger size than a product with less number of features. **Project Estimation techniques**: Estimation of various project parameters is a basic project planning activity. The important project parameters that are estimated include: project size, effort required to develop the software, project duration, and cost. These estimates notonly help in quoting the project cost to the customer, but are also useful in resource planning and scheduling. There are three broad categories of estimation techniques: • Empirical estimation techniques • Heuristic techniques • Analytical estimation techniques., / Empirical Estimation Techniques :Empirical estimation techniques are based on making an educated guess of the project parameters. While using this technique, prior experience with development of similar products is helpful. Although empirical estimation techniques are based on common sense, different activities involved in estimation have been formalized over the years. Two popular empirical estimation techniques are: Expert judgment technique and Delphi cost estimation. • Expert Judgment Technique: Expert judgment is one of the most widely used estimation techniques. In this approach, an expert makes an educated guess of the problem size after analyzing the problem thoroughly. Usually, the expert estimates the cost of the different components (i.e. modules or subsystems) of the system and then combines them to arrive at the overall estimate. However, this technique is subject to human errors and individual bias. Also, it is possible that the expert may overlook some factors inadvertently. • Delphi cost estimation : Delphi estimation iscarried out by a team comprising of a group of experts and a coordinator. In this approach, the coordinator provides each estimator with a copy of the software requirements specification (SRS) document and a form for recording his cost estimate. Estimators complete their individual estimates anonymously and submit to the coordinator. In their estimates, the estimators mention any unusual characteristic of the product which has influenced his estimation. The coordinator prepares and distributes the summary of the responses of all the estimators, and includes any unusual rationale noted by any of the estimators. Based on this summary, the estimators re-estimate. This process is iterated for several rounds. However, no discussion among the estimators is allowed during the entire estimation process. Heuristic Techniques :Heuristic techniques assume that the relationships among the different project parameters can be modeled using suitable mathematical expressions. Once thebasic (independent) parameters are known, the other (dependent) parameters can be easily determined by substituting the value of the basic parameters in the mathematical expression. Analytical Estimation Techniques: Analytical estimation techniques derive the required results starting with basic assumptions regarding the project. Thus, unlike empirical and heuristic techniques, analytical techniques do have scientific basis. **Q18. What is a formal technique ? Explain operational semantics of a formal metho**d A formal specification language consists of two sets syn and sem, and a relation sat between them. The set syn is called the syntactic domain, the set sem is called the semantic domain, and the relation sat is called the satisfaction relation. For a given specification syn, and model of the system sem, if sat(syn, sem) as shown in fig.34.5, then syn is said to be the specification of sem, and sem is said to be the specificand of syn. Syntactic Domains The syntactic domain of a formal specification language consists of an alphabet of symbols and set of formation rules to construct well-formed formulae from the alphabet. The well-formed formulae are used to specify a system. Semantic Domains Formal techniques can have considerably different semantic domains. Abstract data type specification languages are used to specify algebras, theories, and programs. Programming languages are used to specify functions from input to output values. Concurrent and distributed system specification languages are used to specify state sequences, event sequences, statetransition sequences, synchronization trees, partial orders, state machines, etc. Satisfaction RelationIt is important to determine whether an element of the semantic domain satisfies the specifications. This satisfaction is determined by using a homomorphism known as semantic abstraction function. The semantic abstraction function maps the elements of the semantic domain into equivalent classes. There can be different specifications describing different aspects of a system model, possibly using different specification languages. Some of these specifications describe the system's behaviour and the others describe the system's structure. Consequently, two broad classes of semantic abstraction functions are defined: those that preserve a system's behaviour and those that preserve a system's structure. **Operational Semantics:** Informally, the operational semantics of a formal method is the way computations are represented. There are different types of operational semantics according to what ismeant by a single run of the system and how the runs are grouped together to describe the behaviour of the system. Some commonly used operational semantics are as follows: • Linear Semantics In this approach, a run of a system is described by a sequence (possibly infinite) of events or states. The concurrent activities of the system are represented by non-deterministic inter-leavings of the automatic actions. • Branching SemanticsIn this approach, the behaviour of a system is represented by a directed graph as shown in the fig. The nodes of the graph represent the possible states in the evolution of a system. The descendants of each node of the graph represent the states which can be generated by any of the atomic actions enabled at that state. • Maximally Parallel Semantics In this approach, all the concurrent actions enabled at any state are assumed to be taken together. This is again not a natural model of concurrency since it implicitly assumes the availability of all the required computational resources. • Partial Order Semantics Under this view, the semantics ascribed to a system is a structure of states satisfying a partial order relation among the states (events). The partial order represents a precedence ordering among events, and constrains some events to occur only after some other events have occurred; while the occurrence of other events (called concurrent events) is considered to be incomparable. **Q20. What is the primary goal of coding phase ?** Ans:To translate the design of system into a computer language format: The coding is the process of transforming the design of a system into a computer language format, which can be executed by a computer and that perform tasks as specified by the design of operation during the design phase.To reduce the cost of later phases: The cost of testing and maintenance can be significantly reduced with efficient coding.Making the program more readable: Program should be easy to read and understand. It increases code understanding having readability and understandability as a clear objective of the coding activity can itself help in producing more maintainable software.