WINTER – 2024 EXAMINATION
Model Answer – Only for the Use of RAC Assessors

Subject: Software Engineering **Subject Code:** **22413**

Important Instructions to examiners:

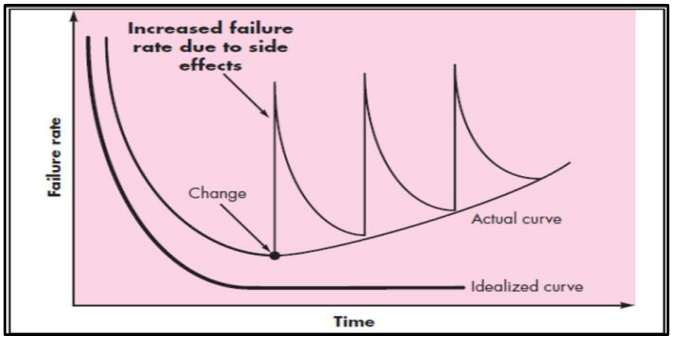
1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling errors should not be given more Importance (Not applicable for subject English and Communication Skills).
4) While assessing figures, examiner may give credit for principal components indicated in the figure. The figures drawn by candidate and model answer may vary. The examiner may give credit for any equivalent figure drawn.
5) Credits may be given step wise for numerical problems. In some cases, the assumed constant values may vary and there may be some difference in the candidate's answers and model answer.
6) In case of some questions credit may be given by judgement on part of examiner of relevant answer based on candidate's understanding.
7) For programming language papers, credit may be given to any other program based on equivalent concept.
8) As per the policy decision of Maharashtra State Government, teaching in English/Marathi and Bilingual (English + Marathi) medium is introduced at first year of AICTE diploma Programme from academic year 2021-2022. Hence if the students in first year (first and second semesters) write answers in Marathi or bilingual language (English +Marathi), the Examiner shall consider the same and assess the answer based on matching of concepts with model answer.

Q. No	Sub Q.N.	Answer	Marking Scheme
1.	**a) Ans.**	**Attempt any FIVE of the following:** **Define Software Engineering** ***Note: Any other relevant definition shall be considered*** Application of **Systematic, disciplined, quantifiable,** approach to the development, operation, and maintenance of software as well as study of these approaches.	**10M** **2M** ***Correct definition 2M***
	b) Ans.	**List types of Software (Any four)** Types of software are : • System software • Application Software • Scientific software • Embedded software • Product line software • Web application software • Artificial Intelligence software	**2M** ***Listing of any four types ½ M each***

	c) Ans.	**State need of SRS** The need of SRS document is to provide 1. A detailed overview of software product, its parameters and goals. 2. The description regarding the project's target audience and its user interface hardware and software requirements. 3. How client, team and audience see the product and its functionality.	**2M** ***Correct needs 2M***
	d) Ans.	**State classification of risk** Types of risks are: 1. Generic risk 2. Product specific risk 3. Schedule / Time-Related / Delivery Related Planning Risks 4. Budget / Financial Risks 5. Operational / Procedural Risks 6. Technical / Functional / Performance Risks 7. Other Unavoidable Risks	**2M** ***Correct Classification of any four ½ M each***
	e) Ans.	**State different types of estimations in Software Projects.** Different types of estimations in Software Projects are : 1. Cost Estimation i) COCOMO ii) COCOMO-II 2. Size estimation i) Function Points ii) Lines Of Code 3. Budget estimation 4. Time estimation	**2M** ***Any 4 types of estimations ½M each***
	f) Ans.	**Differentiate quality control and quality assurances (Any two)** **Quality Assurance (QA)** / **Quality Control (QC)** It is a procedure that focuses on providing assurance that quality requested will be achieved / It is a procedure that focuses on fulfilling the quality requested. QA aims to prevent the defect / QC aims to identify and fix defects	**2M** ***Any two correct differences 1M each***

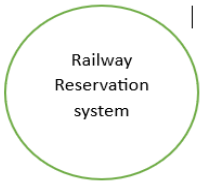
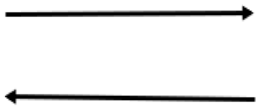
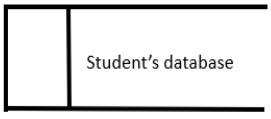
		It is a method to manage the quality- Verification / It is a method to validate the quality-Validation It does not involve executing the program / It always involves executing a program It is a preventive technique. / It is a corrective technique.	
	g) Ans.	**State the concept of Devops**. DevOps is a software development approach that emphasizes collaboration and communication between development (Dev) and operations (Ops) teams. It aims to shorten the software development lifecycle and improve the quality and reliability of software releases. DevOps will remove the "siloed" conditions between the development team and operations team. In many cases these two teams will work together for the entire application lifecycle, from development and test to deployment to operations, and develop a range of skills not limited to a single function. Teams in charge of security and quality assurance may also integrate more closely with development and operation	**2M** ***Correct concept 2M***
2.	**a) Ans.**	**Attempt any <u>THREE</u> of the following:** **State and describe characteristics of Software Engineering** **Characteristics of software are :** 1. Software is developed or engineered; it is not manufactured in the classical sense. Although some similarities exist between software development and hardware manufacture, the two activities are fundamentally different. In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are non-existent (or easily corrected) for software. Both activities are dependent on people, but the relationship between people applied and work accomplished is entirely different. Software costs are concentrated in engineering. This means that software projects cannot be managed as if they were manufacturing projects.	**12M** **4M** ***Description of software characteristics 4M***

2. Software doesn't "wear out."



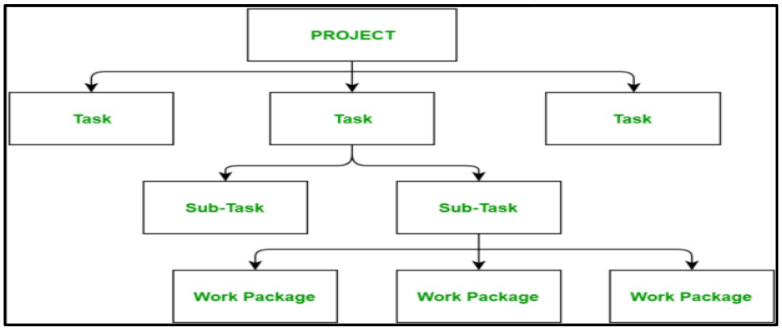
The idealized curve as shown in above figure is a gross oversimplification of actual failure models for software. However, the implication is clear—software doesn't wear out. But it does deteriorate! This contradiction can best be explained by considering the "actual curve" shown in Figure. During its life, software will undergo change (maintenance). As changes are made, it is likely that some new defects will be introduced, causing the failure rate curve to spike as shown in Figure. Before the curve can return to the original steady-state failure rate, another change is requested, causing the curve to spike again. Slowly, the minimum failure rate level begins to rise—the software is deteriorating due to change.

3. Although the industry is moving toward component-based construction, most software continues to be custom built.
The reusable components have been created so that the engineer can concentrate on the truly innovative elements of a design, that is, the parts of the design that represent something new. In the software world, it is something that has only begun to be achieved on a broad scale. A software component should be designed and implemented so that it can be reused in many different programs A software component should be designed and implemented so that it can be reused in many different programs. Modern reusable components encapsulate both data and the processing that is applied to the data, enabling the software engineer to create new applications from reusable parts. For example, today's interactive user interfaces are built with reusable components that enable the creation of graphics windows, pull-down menus, and a wide variety of interaction mechanisms.

	b) Ans.	**Describe symbols used in DFD** 1. Circle: A circle (bubble) shows a process that transforms data inputs into data outputs. For example: a process of Railway reservation system can be indicates as follows:  2. Data Flow: A curved line or straight line with arrow shows the flow of data into or out of a process or data store.  3. Data Store: A set of parallel lines shows a place for the collection of data items. A data store indicates that the data is stored which can be used at a later stage or by the other processes in a different order. The data store can have an element or group of elements. 4. Source or Sink: Source or Sink is an external entity and acts as a source of system inputs or sink of system outputs. This is represented by a rectangle in which the name of the entity is written Eg :  OR 	**4M** ***Any four Correct symbols with explanation 1M each***

	c) Ans.	**Describe 4 P's of Management Spectrum** The management Spectrum: 4p's Effective software project management focuses on the four P's: people, product, process, and project. **1. People:** 1. The "people factor" is so important that the Software Engineering Institute has developed a People Capability Maturity Model (People CMM) to continually improve its ability to attract, develop, motivate, organize, and retain the workforce needed to accomplish its strategic business objectives. 2. The people capability maturity model defines the following key practice areas for software people: a. Staffing b. communication and coordination c. work environment d. performance management e. Training, compensation, competency analysis and development, career development, workgroup development, team/culture development and others. 3. Organizations that achieve high levels of People-CMM maturity have higher likelihood of implementing effective software project management practices. **2. Product:** 1. Before a project can be planned, product objectives and scope should be established, alternative solutions should be considered and technical and management constraints should be identified . 2. Without this information, it is impossible to define reasonable (and accurate) estimates of the cost, an effective assessment of risk, a realistic breakdown of project tasks, or a manageable project schedule that provides a meaningful indication of progress. 3. Objectives identify the overall goals for the product (from the stakeholders' points of view) without considering how these goals will be achieved. 4. Scope identifies the primary data, functions, and behaviors that characterize the product 5. The alternatives enable managers and practitioners to select a "best" approach, given the constraints imposed by delivery deadlines, budgetary restrictions, personnel availability, technical interfaces, and other factors.	**4M** ***Explanation of 4 P's 1M each***

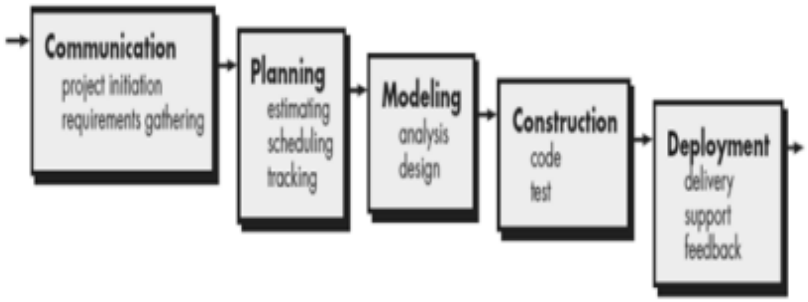
		3. Process: 1. A software process provides the framework from which a comprehensive plan for software development can be established. 2. A small number of framework activities are applicable to all software projects, regardless of their size or complexity. 3. A number of different task sets—tasks, milestones, work products, and quality assurance points enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team. 4. Finally, umbrella activities—such as software quality assurance, software configuration management, and measurement occur throughout the process. **4. Project:** 1. To manage complexity, we conduct planned and controlled software projects. 2. The success rate for present-day software projects may have improved but our project failure rate remains much higher than it should be. 3. To avoid project failure, a software project manager and the software engineers who build the product must avoid a set of common warning signs, understand the critical success factors that lead to good project management, and develop a common-sense approach for planning, monitoring, and controlling the project.	
	d) Ans.	**Explain work breakdown structure with suitable diagram.** ***Note: Any other relevant example shall be considered.*** A Work Breakdown Structure includes dividing a large and complex project into simpler, manageable, and independent tasks. The root of this tree (structure) is labeled by the Project name itself. For constructing a work breakdown structure, each node is recursively decomposed into smaller sub-activities, until at the leaf level, the activities become undividable and independent. It follows a Top-Down approach. Steps Work Breakdown Structure: Step 1: Identify the major activities of the project. Step 2: Identify the sub-activities of the major activities. Step 3: Repeat till undividable, simple, and independent activities are created.	**4M** ***Working of structure 2M*** ***Example with diagram 2M***



Construction of Work Breakdown Structure

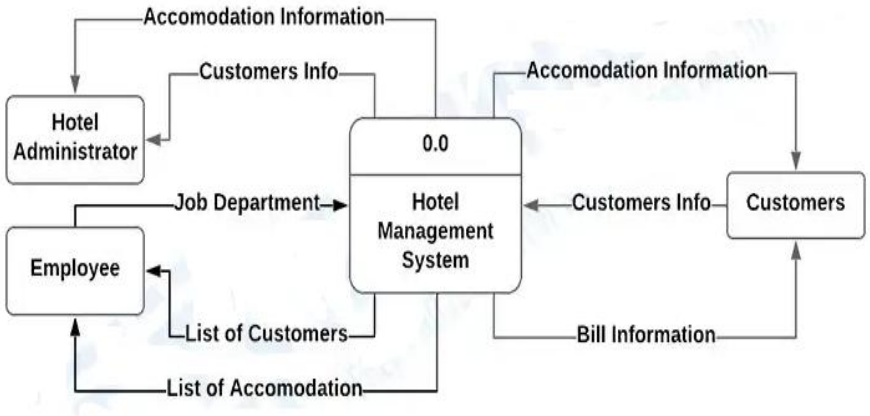
1. Firstly, the project managers and top level management identifies the main deliverables of the project.
2. After this important step, these main deliverables are broke down into smaller higher-level tasks and this complete process is done recursively to produce much smaller independent tasks.
3. It depends on the project manager and team that upto which level of detail they want to break down their project.
4. Generally the lowest level tasks are the most simplest and independent tasks and takes less than two weeks worth of work.
5. Hence, there is no rule for upto which level we may build the work breakdown structure of the project as it totally depends upon the type of project we are working on and the management of the company.
6. The efficiency and success of the whole project majorly depends on the quality of the Work Breakdown Structure of the project and hence, it implies its importance.

Example:

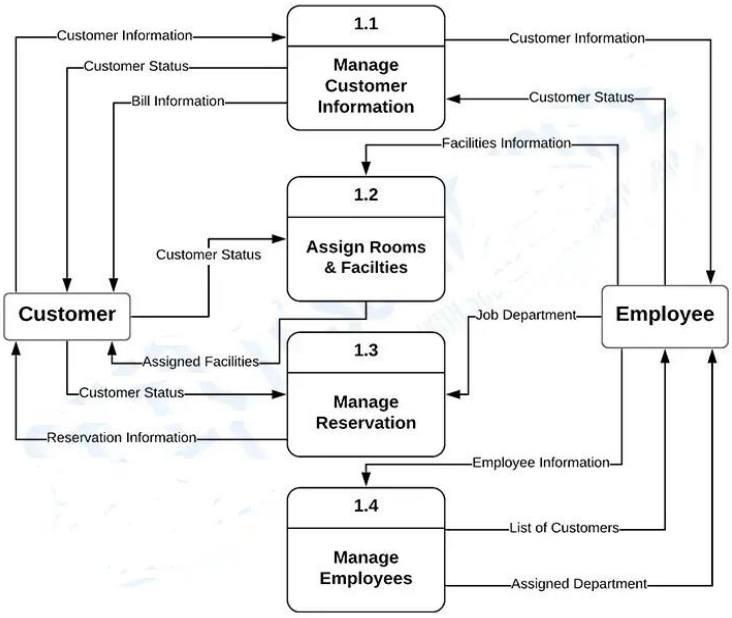
3.	**a)** **Ans.**	**Attempt any THREE of the following:** **With the help of a suitable diagram describe waterfall model with any two advantages and disadvantages.**  Fig: The Waterfall Model **The Waterfall Model:** The waterfall model is a traditional method, sometimes called the classic life cycle. This is one of the initial models. As the figure implies, stages are cascaded and shall be developed one after the other. It suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through communication, planning, modeling construction and deployment. In other words, one stage should be completed before the other begins. Hence, when all the requirements are elicited by the customer, analyzed for completeness and consistency, documented as per requirements, the development and design activities commence. One of the main needs of this model is the user ‘s explicit prescription of complete requirements at the start of development. For developers it is useful to lay out what they need to do at the initial stages. Its simplicity makes it easy to explain to customers who may not be aware of the software development process. It makes explicit with intermediate products to begin at every stage of development.	**12M** **4M** ***Correct description of Waterfall Model with diagram 2M,*** ***Any two advantages ½M each*** ***Any two disadvantages ½M each***

		One of the biggest limitations is it does not reflect the way code is really developed. The problem is well understood but the software is developed with a great deal of iteration. Often this is a solution to a problem which was not solved earlier and hence software developers shall have extensive experience of developing such applications as neither the user nor the developers are aware of the key factors affecting the desired out come and the time needed. Hence at times the software development process may remain uncontrolled. Today software work is fast paced and subject to a never-ending stream of changes in features, functions and information content. Waterfall model is inappropriate for such work. This model is useful in situations where the requirements are fixed and work proceeds to completion in a linear manner. **Advantages of waterfall model:** 1. This model is simple and easy to understand and use. 2. It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process. 3. In this model phases are processed and completed one at a time. Phases do not overlap. 4. The waterfall model works well for smaller projects where requirements are very well understood. **Disadvantages of waterfall model:** 1. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage. 2. No working software is produced until late during the life cycle. 3. High amounts of risk and uncertainty. 4. Not a good model for complex and object-oriented projects. 5. Poor model for long and ongoing projects. 6. Not suitable for the projects where requirements are at a moderate to high risk of changing.	

	b) **Ans.**	**Describe communication practices. (Any four)** **Communication Practices:** 1. Listen • Try to focus on the speaker's words, rather than formulating your response to those words. • Ask for clarification if something is unclear but avoid constant interruptions. • Never become contentious in your words or actions (e.g., rolling your eyes or shaking your head) as a person is talking. 2. Prepare before you communicate • Spend the time to understand the problem before you meet with others. If necessary, perform some research to understand business domain. • If you have responsibility for conducting a meeting, prepare an agenda in advance of the meeting. 3. Someone should facilitate the activity • Every communication meeting should have a leader (a facilitator) • To keep the conversation moving in a productive direction. • To mediate any conflict that does occur, and • To ensure that other principles are followed. 4. Face-to-face communication is best • It usually works better when some other representation of the relevant information is present. • For example, a participant may create a drawing /document that serves as a focus for discussion. 5. Take notes and document decisions Someone participating in communication should serve as a recorder and write down all important points and decisions. 6. Strive for collaboration • Collaboration occurs when the collective knowledge of members of the team is used to describe product or system functions or features. • Each small collaboration builds trust among team members and creates a common goal for the team. 7. Stay focused; modularize your discussion • The more people involved in any communication, the more likely that discussion will bounce from one topic to the next.	**4M** ***Any four correct communication practices with description 1M each***

		• The facilitator should keep the conversation modular, leaving one topic only after it has been resolved. 8. If something is unclear, draw a picture • Verbal communication goes only so far. • A sketch or drawing can often provide clarity when words fail to do the job. 9. a) Once you agree to something, move on. b) If you can't agree to something, move on. c) If a feature or function is unclear and cannot be clarified at the moment, move on. • The people who participate in communication should recognize that many topics require discussion and that moving on is sometimes the best way to achieve communication agility. 10.Negotiation is not a contest or a game • It works best when both parties win. • There are many instances in which you and other stakeholders must negotiate functions and features, priorities, and delivery dates. • If the team has collaborated well, all parties have a common goal. Still, negotiation will demand compromise from all parties.	
	c) Ans.	Draw DFD-0 and DFD-1 for Hotel Management System. **DFD-0 for Hotel Management System**  	**4M** ***Hotel Management System*** ***DFD 0-2M*** ***DFD 1-2M***

DFD-1 for Hotel Management System



d) Ans. **Describe function point metrics for size estimation.** **4M**

Correct Description of function point metrics 2M

UFP formula 1M

TCF formula 1M

- 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 a smaller number of features.
- Each function when invoked reads some input data and transforms it to the corresponding output data.
- For example, the issue book feature of Library Automation Software takes the name of the book as input and displays its location and the number of copies available.
- Thus, a computation of the number of input and the output data values to a system gives some indication of the number of functions supported by the system.
- In addition to the number of basic functions that software performs, the size is also dependent on the number of files and the number of interfaces.

		• Besides using the number of input and output data values, function point metric computes the size of a software product (in units of functions points or FPs) using three other characteristics of the product as shown in the following expression. • The size of a product in function points (FP) can be expressed as the weighted sum of these five problem characteristics. • The weights associated with the five characteristics were proposed empirically and validated by the observations over many projects. • Function point is computed in two steps. The first step is to compute the unadjusted function point (UFP). UFP = (Number of inputs) *4 + (Number of outputs) *5 + (Number of inquiries) *4 + (Number of files)*10 + (Number of interfaces)*10 • Once the unadjusted function point (UFP) is computed, the technical complexity factor (TCF) is computed next. • TCF refines the UFP measure by considering fourteen other factors such as high transaction rates, throughput, and response time requirements, etc. • Each of these 14 factors is assigned from 0 (not present or no influence) to 6 (strong influence). • The resulting numbers are summed, yielding the total degree of influence (DI). • Now, TCF is computed as TCF=(0.65+0.01*DI) As DI can vary from 0 to 70, TCF can vary from 0.65 to 1.35. Finally, FP=UFP*TCF.	
4.	**a)** **Ans.**	**Attempt any <u>THREE</u> of the following:** **Describe any four characteristics of Agile process.** **Characteristics of Agile process:** • **Modularity** Modularity is a key element of any good process. Modularity allows a process to be broken into components called activities. A software development process prescribes a set of activities capable of transforming the vision of the software system into reality. Activities are used in the agile software process like a good tool. They are to be	**12M** **4M** ***Any four correct characteristics with description 1M each***

		wielded by software craftsmen who know the proper circumstances for their use. They are not utilized to create a production-line atmosphere for manufacturing software. • **Iterative** Agile software processes acknowledge that we get things wrong before we get them right. Therefore, they focus on short cycles. Within each cycle, a certain set of activities is completed. These cycles will be started and completed in a matter of weeks. However, a single cycle called iteration will probably not be enough to get the element 100% correct. • **Time-Bound** Iterations become the perfect unit for planning the software development project. One can set time limits on each iteration and schedule them accordingly. Chances are that the designer will not (unless the process contains very few activities) schedule all the activities the process does in a single iteration. Instead, we will only attempt those activities necessary to achieve the goals set out at the beginning of the iteration. Functionality may be reduced, or activities may be rescheduled if they cannot be completed within the allotted period. • **Parsimony** Agile Process is more than a traditional software development process with some time constraints. Attempting to create impossible deadlines under a process not suited for rapid delivery puts the onus on the software developers. This leads to burnout and poor quality. Instead, agile software processes focus on parsimony. That is, they require a minimal number of activities necessary to mitigate risks and achieve their goals. • **Adaptive** During an iteration, new risks may be exposed which require some activities that were not planned. The agile process adapts the process to attack these newfound risks. If the goal cannot be achieved using the activities planned during the iteration, new activities can be added to allow the goal to be reached. Similarly, activities may be discarded if the risks turn out to be ungrounded. • **Incremental** An agile process does not try to build the entire system at once. Instead, it partitions the non trivial system into increments which may be developed in parallel, at different times, and at different rates. We	

		unit test each increment independently. When an increment is completed and tested, it is integrated into the system. • **Convergent** Convergence states that we are actively attacking all of the risks worth attacking. As a result, the system becomes closer to the reality that we seek with each iteration. As risks are being pro actively attacked, the system is being delivered in increments. We are doing everything within our power to ensure success in the most rapid fashion. • **People-Oriented** Agile processes favor people over process and technology. They evolve through adaptation in an organic manner. Developers that are empowered raise their productivity, quality, and performance. • **Collaborative** Agile processes foster communication among team members. Communication is a vital part of any software development project. When a project is developed in pieces, understanding how the pieces fit together is vital to creating the finished product. There is more integration than simple communication. Quickly integrating a large project while increments are being developed in parallel requires collaboration.	
	b) **Ans.**	**Describe core principles of Software Engineering.** **Core principles of Software Engineering:** Core principles of Software Engineering Practices: 1. Reason it all exists. • The software system exists to provide value for the user. • Before specifying the problem, the requirement and the specifications have to be laid down. • The hardware and the software platform to be decided for implementation. 2. Keep it simple stupid • The terms and the design used for the development of the project should be kept simple and easily understandable. • All the terms used should be easy to facilitate the basic concept of the project. 3.Maintain the vision • A clear vision is important for the development of software. • Compromising the architectural vision of the project weakens the development of the software.	**4M** ***Description of 4 software core principles 1M each***

		• The developer should hold the vision and ensure the successful development and deployment of the software. 4.What you reproduce, someone else will have to consume. (Implement knowing someone else will have to understand what you are doing) • Always specify, design, and implement knowing that someone else is going to understand what is being developed. • Customers for product development are very large. • Design the data structure and the implementation, keeping implementation in mind and the end user. • Code with the concern that the product must be implemented and maintained by the end user. 5. Be open to the future • The system designed today should be adaptable to the development and changes in the future at a low cost. • There should not be many changes to the software to adapt to the new changes in future development. 6. Plan ahead for reuse. • The design and specifications should be developed in such a way that they can be reused for other implementations. • The code and the design should be well documented for the use in future. 7. Think! • Before designing and implementing, a proper thought should be to the result. • Proper data structure and the design and implementation strategy should be developed if the software needs modification in the future.	
	c) Ans.	**Describe RMMM strategy.** Risk Mitigation, Monitoring and Management (RMMM) plan: • A risk management strategy can be included in the software project plan, or the risk management steps can be organized into a separate Risk Mitigation, Monitoring and Management Plan. • The RMMM plan documents all work performed as part of risk analysis and is used by the project manager as part of the overall project plan. • Once RMMM has been documented and the project has begun, risk mitigation and monitoring steps commence.	**4M** ***Correct Description of RMMM strategy 4M***

- Risk mitigation is a problem avoidance activity.
- Risk monitoring is a project tracking activity with three primary objectives:
  1) To assess whether predicted risks do, in fact, occur.
  2) To ensure that risk aversion steps defined for the risk are being properly applied.
  3) To collect information that can be used for future risk analysis.
- In many cases, the problems that occur during a project can be traced to more than one risk.
- Another job of risk monitoring is to attempt to allocate origin (what risk(s) caused which problems throughout the project).
- An effective strategy must consider three issues:
  1) Risk avoidance
  2) Risk monitoring
  3) Risk management and contingency planning.
- If a software team adopts a proactive approach to risk, avoidance is always the best strategy. This is achieved by developing a plan for risk mitigation.

Consider example of Staff Turnover risk:

To mitigate this risk, project management must develop a strategy for reducing turn over. Among the possible steps to be taken are:

1) Meet with current staff to determine causes for turnover (e.g., poor working conditions, low pay, and competitive job market).
2) Mitigate those causes that are under our control before the project starts.
3) Once the project commences, assume turnover will occur and develop techniques to ensure continuity when people leave.
4) Organize project teams so that information about each development activity is widely dispersed.
5) Define documentation standards and establish mechanisms to be sure that documents are developed in a timely manner.
6) Conduct peer reviews of all work (so that more than one person is "up to speed).
7) Assign a backup staff member for every critical technologist. As the project proceeds, risk monitoring activities commence.

The project manager monitors factors that may provide an indication of whether the risk is becoming more or less likely.

		In the case of high staff turnover, the following factors can be monitored: 1) General attitude of team members based on project pressures. 2) The degree to which the team has jelled. 3) Interpersonal relationships among team members. 4) Potential problems with compensation and benefits. 5) The availability of jobs within the company and outside it. In addition to monitoring these factors, the project manager should monitor the effectiveness of risk mitigation steps. RMMM steps incur additional project cost. Part of risk management, therefore, is to evaluate when the benefits accrued by the RMMM steps are outweighed by the costs associated with implementing them. Project planner performs a classic cost/benefit analysis.	
	d) Ans.	**State and describe project cost estimation approach. (Any two)** Project cost estimation approaches: 1) Heuristic Estimation Approach 2) Analytical Estimation Approach 3) Empirical Estimation Approach **1) Heuristic Estimation Approach** Heuristic techniques assume that the relationships among the different projects parameters can be modeled using suitable mathematical expressions. Once the basic (independent) parameters are known, the other (dependent) parameters can be easily determined by substituting the value of the basic parameters in the mathematical expression. Different heuristic estimation models can be divided into the following two classes: • Single variable model • Multi variable model Single variable estimation models provide a means to estimate the desired characteristics of a problem, using some previously estimated basic (independent) characteristics of the software product such as its size. A single variable estimation model takes the following form: Estimated Parameter = $c_1 * e_1^{d_1}$	**4M** ***Naming of project cost estimation approaches 1M,*** ***Description of any two approaches 3M***

In the above expression, e is the characteristic of the software which has already been estimated (independent variable). Estimated Parameter is the dependent parameter to be estimated. The dependent parameters to be estimated could be effort, project duration, staff size, etc. c1 and d1 are constants. The values of the constants c1 andd1 are usually determined using data collected from past projects (historical data).

The basic COCOMO model is an example of single variable cost estimation model.

A multivariable cost estimation model takes the following form:

$$\text{Estimated Resource} = c_1 * e_1^{d_1} + c_2 * e_2^{d_2} + \ldots$$

Where e1, e2, … are the basic (independent) characteristics of the software already estimated, and c1, c2, d1, d2, … are constants.

2) Analytical

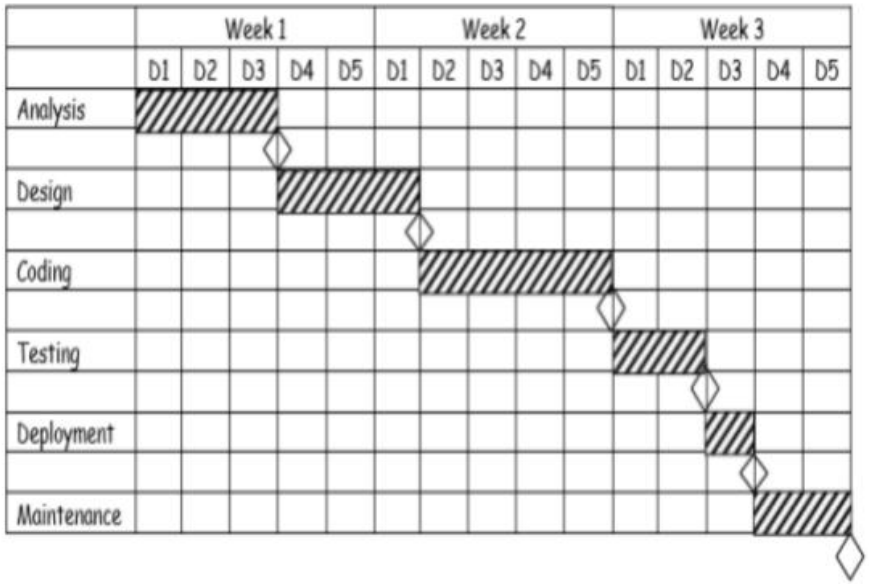
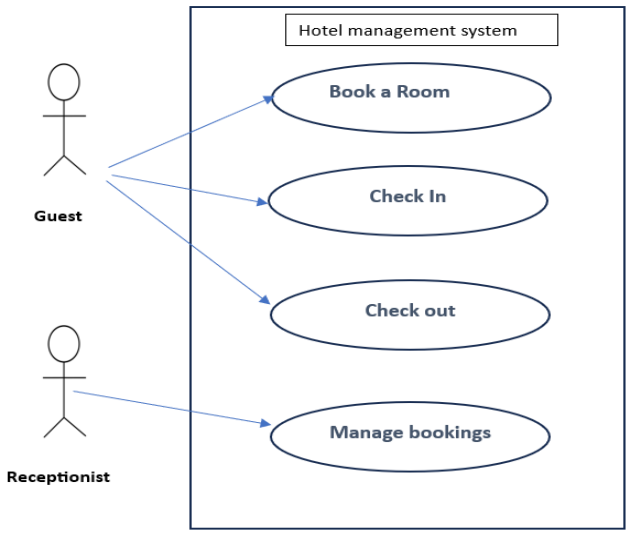
Analytical estimation techniques derive the required results starting with basic assumptions regarding the project. Thus, unlike empirical and heuristic techniques, analytical techniques do have scientific basis. Halstead's software science is an example of an analytical technique. Halstead's software science can be used to derive some interesting results, starting with a few simple assumptions. Halstead's software science is especially useful for estimating software maintenance efforts. In fact, it outperforms both empirical and heuristic techniques when used for predicting software maintenance efforts. Halstead's Software Science – An Analytical Technique Halstead's software science is an analytical technique to measure size, development effort, and development cost of software products.

Halstead used a few primitive program parameters to develop expressions for overall program length, potential minimum value, actual volume, effort, and development time. For a given program, let:

- η1 be the number of unique operators used in the program,
- η2 be the number of unique operands used in the program,
- N1 be the total number of operators used in the program,
- N2 be the total number of operands used in the program.

		Example: Let us consider the following C program:
		main( )
		{
		int a, b, c, avg;
		scanf("%d %d %d", &a, &b, &c);
		avg = (a+b+c)/3;
		printf("avg = %d", avg);
		}
		
		The unique operators are: main, (), {}, int, scanf, &, ", ", =, +, /,printf
		The unique operands are: a, b, c, &a, &b, &c, a+b+c, avg, 3, "%d %d%d", "avg = %d"
		
		Therefore, η1 = 12, η2 = 11
		
		Estimated Length = (12*log12 + 11*log11) = (12*3.58 + 11*3.45) = (43+38) = 81
		Volume = Length*log(23) = 81*4.52 = 366
		
		3) Empirical
		Empirical estimation is a technique or model in which empirically derived formulae are used for predicting the data that are a required and essential part of the software project planning step.
		These techniques are usually based on the data that is collected previously from a project and based on some guesses, prior experience with the development of similar types of projects, and assumptions.
		It uses the size of the software to estimate the effort. In this technique, an educated guess of project parameters is made. Hence, these models are based on common sense. However, as there are many activities involved in empirical estimation techniques, this technique is formalized.	

	e)	**Prepare Time line chart for ATM machine.**	**4M**

	Ans.	**Timeline chart:**  **Fig: Timeline chart for ATM machine**	*Correct Time line chart 4M*
5.	**a)** **Ans.**	**Attempt any TWO of the following:** **a) Sketch Use-Case diagram for Hotel Management System with minimum two actors and four use cases.** ***Note: Any other relevant diagram shall be considered***  	**12M** **6M** ***Each use case 1M Each actor and interaction 1M***

b) Ans.

Differentiate between Black Box Testing and White Box Testing. (Six points) — **6M**

Any six correct differences 1M each

Sr No	White box testing	Black Box Testing
1	The tester needs to have the knowledge of internal code or program	This technique is used to test the software without the knowledge of internal code or program.
2	It aims at testing the structure of the item being tested.	It aims at testing the functionality of the software.
3	It is also called structural testing, clear box testing, code-based testing, or glass box testing.	It also known as data driven, dark box testing, opaque box and functional testing.
4	Testing is best suited for a lower level of testing like Unit Testing, Integration testing.	This type of testing is ideal for higher levels of testing like System Testing, Acceptance testing.
5	Statement Coverage, Branch coverage, and Path coverage are White Box testing technique.	Equivalence partitioning, Boundary value analysis are Black Box testing technique
6	Can be based on detailed design documents	Can be based on Requirement specification document

c) Ans.

Use COCOMO model to calculate: — **6M**

Each correct calculation 1½M

i) Efforts
ii) Development Time
iii) Average staff size
iv) Productivity.
If estimation size of product is 400 KLOC using organic model

In the basic COCOMO Model which is used to estimate the development time of software

Effort: E = a * [(KLOC)^b] = X person-months

Development time :D= c *[Effort^d] = Y month

Productivity :P= {KLOC}/{E} KLOC/person-month

Staff Size :S= {E}/{D} persons

where a , b , c , d are constants derived from the table below depending on the type of your software project

Project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Organic Mode:

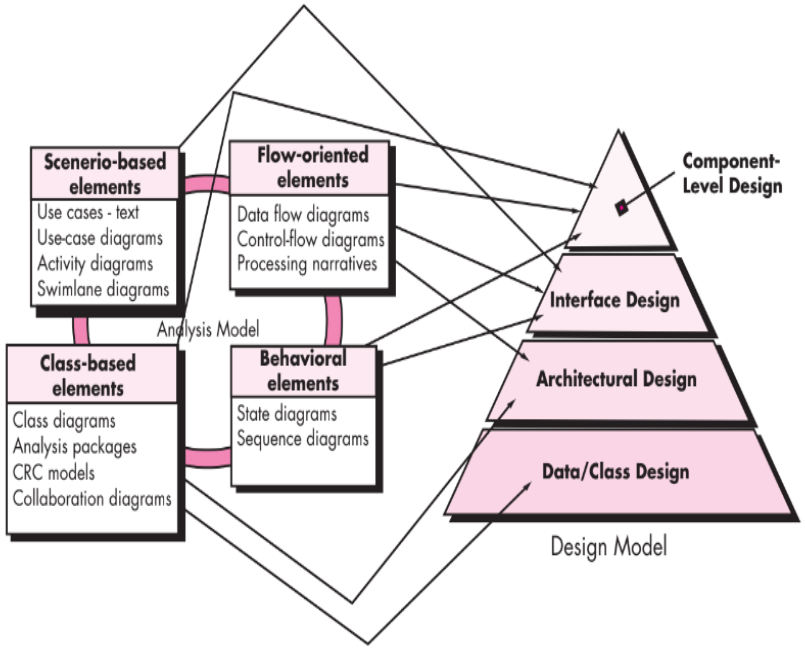
Effort: E = 2.4*[(KLOC)^{1.05}]
E = 2.4 * [(400)^{1.05}]
E = **1295.31**person-months

Development Time: D= 2.5 * [(E)^{0.38}]
D= 2.5 * (1295.31)^{0.38}]
D= **38.07** months = 38months(rounded)

Staff Size: S= {E}/{D}
S= {1295.31}/{38.07}
S = 34.02 = 34 persons (rounded up)

Productivity: P= {KLOC}/{E}
P = {400}/{1295.31}
P= 0.3088 KLOC/person-month

6.		**Attempt any <u>TWO</u> of the following:**	**12M**
	a)	**Draw neat labelled diagram for translation of requirement model into design model.**	**6M**
	Ans.	**Translating the requirement model into design model**	

			***Analysis model Each Block 1M*** ***Design Model 1M*** ***Interaction 1M***
	b) Ans.	**Describe six sigma. State operations in DMADV/DMAIC.** ➢ *Six Sigma* is the most widely used strategy for statistical quality assurance in industry today. ➢ Originally popularized by Motorola in the 1980s, the Six Sigma strategy "is a rigorous and disciplined methodology that uses data and statistical analysis to measure and improve a company's operational performance by identifying and eliminating defects' in manufacturing and service-related processes. The Six Sigma methodology defines **three core steps:** **Define** :*Define* customer requirements and deliverables and project goals via well defined methods of customer communication. **Measure** :*Measure* the existing process and its output to determine current quality performance (collect defect metrics). **Analyze** *:Analyse the* defect metrics and determine the vital few causes. If an existing software process is in place**, but improvement is required**, Six Sigma suggests two additional steps: • **Improve** *: Improve* the process by eliminating the root causes of	**6M** ***Description 2M*** ***DMAIC 2M*** ***DMADV 2M***

		defects. • **Control** *: Control* the process to ensure that future work does not reintroduce the causes of defects. These core and additional steps are sometimes referred to as the **DMAIC** (define, measure, analyze, improve, and control) method. If an organization is developing a software process (rather than improving an existing process), the core steps are augmented as follows: • **Design** *:Design* the process to 1) avoid the root causes of defects and 2) to meet customer requirements. • **Verify** *: Verify* that the process model will avoid defects and meet customer requirements. This variation is sometimes called the **DMADV** (define, measure, analyze, design, and verify) method.	
	c) **Ans.**	**Recognize requirements for following modules of banking software:** **i) Customer Module** **ii) Loan Module** **iii) Account Module.** **Customer Module** 1. **Customer Profile Management**: Store and manage customer details such as name, contact information, address, and identification documents. 2. **Account Management**: Allow customers to open, close, and manage multiple accounts (savings, checking, etc.). 3. **Transaction History**: Maintain a record of all transactions made by the customer. 4. **Customer Support**: Provide support for customer inquiries and issues. 5. **Security**: Implement strong authentication and authorization mechanisms to protect customer data. 6. **KYC Compliance**: Ensure compliance with Know Your Customer (KYC) regulations. **Loan Module** 1. **Loan Application Processing**: Handle the submission, review,	**6M** ***Any two relevant requirements for each module 2M***

		and approval of loan applications. 2. **Credit Scoring**: Assess the credit worthiness of applicants using credit scores and other financial data. 3. **Loan Management**: Track the status of loans, including disbursement, repayment schedules, and defaults. 4. **Interest Calculation**: Calculate interest rates and repayment amounts accurately. 5. **Documentation**: Manage loan-related documents such as agreements, collateral records, and repayment schedules. 6. **Compliance**: Ensure adherence to regulatory requirements and internal policies. **Account Module** 1. **Account Creation and Maintenance**: Enable the creation, modification, and deletion of customer accounts. 2. **Balance Management**: Allow customers to view and manage their account balances. 3. **Transaction Processing**: Process deposits, withdrawals, transfers, and other transactions. 4. **Statement Generation**: Generate periodic account statements for customers. 5. **Interest Calculation**: Calculate interest on savings accounts and other interest-bearing accounts. **6. Security**: Implement measures to prevent unauthorized access and fraud.	