WINTER– 16 EXAMINATION

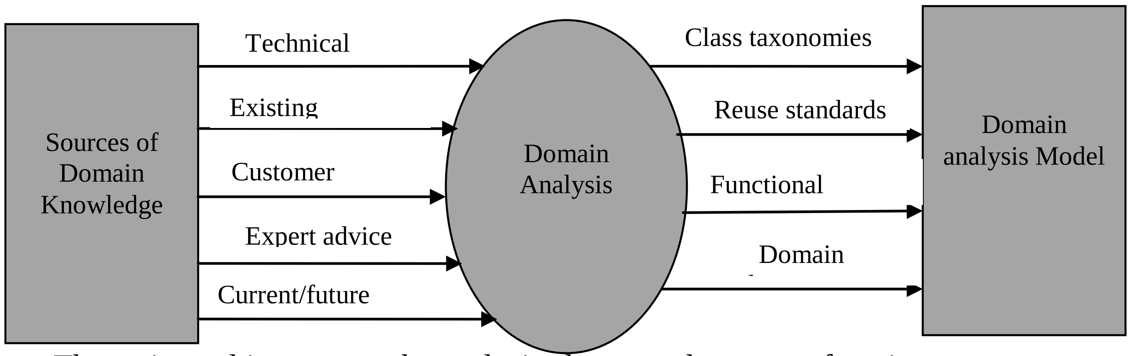
Model Answer Subject Code: 17513

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.

Q. No.	Sub Q. N.	Answer	Marking Scheme
1.	**(A)**	**Answer any THREE of the following :**	**12**
	(a)	**Describe any four categories of software.**	**4M**
	Ans:	**1. System Software:** System Software is a collection of programs written to serve other programs. Some system software (e.g.- compliers, editors, and file management utilities) processes complex, but determinate information structures. Other system applications (e.g.- operating system components, drivers, networking software, telecommunications processors) process largely indeterminate data. In either case, the systems software area is characterized by heavy interaction with computer hardware; heavy usage by multiple users; concurrent operation that requires scheduling, resource sharing, and sophisticated process management; complex data structures; and multiple external interfaces. **2. Application Software:** Application Software consists of standalone programs that solve a specific business need. Applications in this area process business or technical data in a way that facilities business operations or management / technical decision-making. **3. Engineering / Scientific Software:** Formerly characterized by "number crunching" algorithms, engineering and scientific software applications range from astronomy to volcano logy, from automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing. Computer-aided design, system simulation, and other interactive applications have begun to take on real–time and even system software characteristics. **4. Embedded Software:** Embedded Software resides within a product or system and is used to implement and control features and functions for the end-user and for the system itself. Embedded software can perform limited and esoteric functions (e.g. keypad	***(Any 4 Categories: 1 mark per category)***

		control for a microwave oven) or provide significant function and control capability (e.g. digital functions in an automobile such as fuel control, dashboard displays, braking systems, etc.) **5. Product–line Software:** Designed to provide a specific capability for use by many different customers, product–line software can focus on a limited & esoteric market place (e.g. – inventory control products) or address mass consumer markets (e.g. – word processing, spreadsheets, and computer graphics, and multimedia, entertainment, and database management, personal and business financial applications.) **6. Web – applications:** "Web Apps", span a wide array of applications. Web apps are evolving into sophisticated computing environments that not only provide standalone features, computing functions, and content to the end user, but also are integrated with corporate databases and business applications. **7. Artificial Intelligence Software:** AI Software makes use of non–numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis. Applications within this area include robotics, expert systems, pattern recognition (image and voice), artificial neural networks, theorem proving, and game playing.	
	(b)	**State and describe six principles of communication practices.**	**4M**
	Ans:	**Communication Practices** Effective communication among the technical peers, customers and other stakeholders, project managers etc. is among the most challenging activities that confront software engineers. Before customers' requirements can be analyzed, modeled are specified they must be gathered through a communication. Effective communication is among the most challenging activities that you will confront. **Communication Principles are** 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. 2. **Prepare before you communicate:** Speed the time to understand the problem before you meet with others. If necessary, do some research to understand business domain jargon. 3. **Someone should facilitate the activity:** Every communication meeting should have a leader to keep the conversation moving in a productive direction, to mediate any conflict that does occur and to ensure than 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 e.g. A participant may create a drawing or a "straw man" document that serves as a focus for discussion. 5. **Take notes and document decisions:** Things have a way of falling into cracks. Someone participating in the communication should serve as a "recorder" and write down all important points and decisions. 6. **Strive for collaboration:** Collaboration and consensus occur when the collective knowledge of members of the team is used to describe product or system functions or features. Each small collaboration serves to build trust among team members and creates a common goal for the team.	***(List of any 6 Principles: 1 mark; Description: ½ mark each)***

		7. **Stay focused; modularize your discussion:** The more likely involved in any communication, the more likely that discussion will bounce from one topic to next. 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. :** Communication, like any software engineering activity, takes time. Rather than iterating endlessly the people who participates should recognize that many topics require discussion and that "moving on" is sometimes the best 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)	**With neat diagram, describe inputs and output of domain analysis.**	**4M**
	Ans:	Software domain analysis is the identification, analysis and specification of common requirements from a specific application domain, typically for reuse in multiple projects within that application domain. Object-oriented domain analysis is the identification, analysis and specification of common, reusable capabilities within a specific application domain, in terms of common objects, classes, subassemblies, and frameworks. **Input and Output Structure of domain analysis:**  Sources of Domain Knowledge Technical Existing Customer Expert advice Current/future Domain Analysis Class taxonomies Reuse standards Functional Domain Domain analysis Model • The main goal is to create the analysis classes and common functions. • The input consists of the knowledge domain. • The input is based on the technical survey, customer survey and expert advice. • The output domain consists of using the input as the reference and developing the functional models. The role of domain analyst is to discover and define reusable analysis patterns, analysis classes and related information that may be used by many people working on similar but not necessarily the same applications.	***(Diagram: 2 marks Description: 2 marks)***
	(d)	**Write any four features of Agile Software Development approach.** ***{**Any other relevant features can also be consider**}***	**4M**
	Ans:	Agile programming is an approach to project management, typically used in software development. It helps teams react to the instability of building software through	***(Any 4 Features: 1***

incremental, iterative work cycles, known as sprints.

mark each)

Features of the Agile Software Development Approach: The name "agile software process", first originated in Japan. The Japanese faced competitive pressures, and many of their companies, like their American counterparts, promoted cycle-time reduction as the most important characteristic of software process improvement efforts

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 wielded by software craftsman 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.

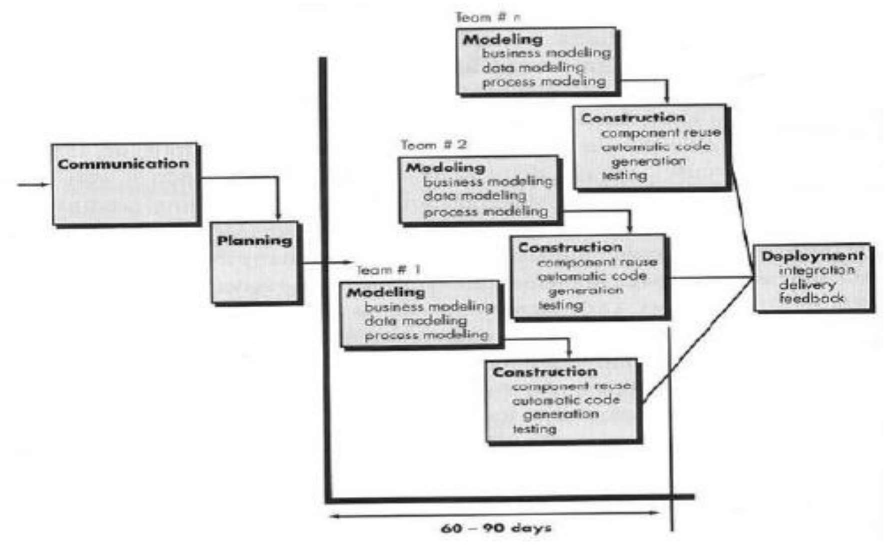
Time-Bound: Iterations become the perfect unit for planning the software development project. We can set time limits (between one and six weeks is normal) on each iteration and schedule them accordingly. Chances are, we will not (unless the process contains very few activities) schedule all of the activities of our process 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 time period.

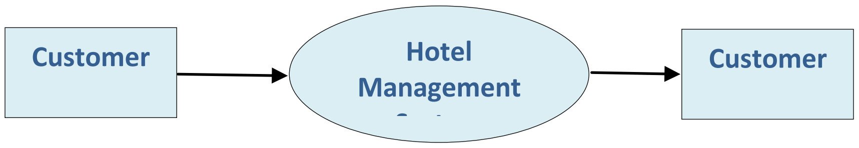
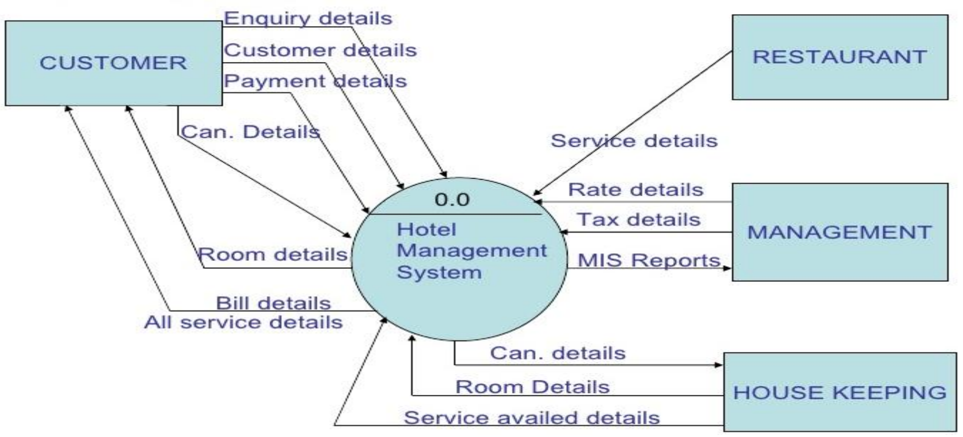
Parsimony: Agile Process is more than a traditional software development process with some time constraints. Attempting to create impossible deadlines under a process not suited for rapid delivery puts the onus on the software developers. This leads to burnout and poor quality Instead, agile software processes focus on parsimony. That is, they require a minimal number of activities necessary to mitigate risks and achieve their goals.

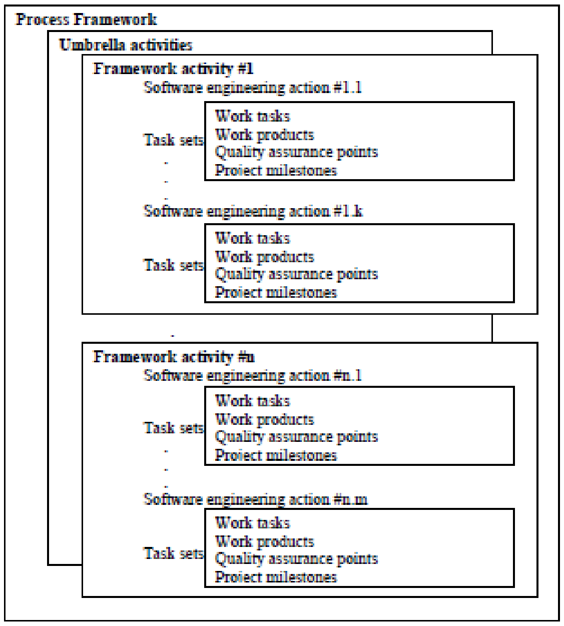
Adaptive: During an iteration, new risks may be exposed which require some activities that were not planned. The agile process adapts the process to attack these new found risks. If the goal cannot be achieved using the activities planned during the iteration, new activities can be added to allow the 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 nontrivial 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 proactively 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 to integration than simple communication. Quickly integrating a large project while increments are being developed in parallel requires collaboration.	
	(B)	**Answer any ONE of the following:**	**6M**
	(a)	**With neat diagram , explain RAD model with its advantages and disadvantages.(2 each)**	**6M**
	Ans:	 Communication Planning Team # 1 Team # 2 Team # n Modeling business modeling data modeling process modeling Construction component reuse automatic code generation testing Deployment integration delivery feedback 60 – 90 days Rapid application on Development (RAD) is a modern software process model that emphasizes a short development cycle. The RAD Model is a "high-speed" adaptation of the waterfall model, in which rapid development is achieved by using a component based construction approach. If requirements are well understood and project scope is considered, the RAD process enables a development team to create a "Fully Functional System" within a very short period of time (e.g. 60 to 90 days). One of the distinct features of RAD model is the possibility of cross life cycle activities which will be assigned to teams, teams #1 to team #n leading to each module getting developed almost simultaneously. This approach is very useful if the business application requirements are modularized as function to be completed by individual teams and finally to integrate into a complete system. As such compared to waterfall model the team will be of larger size to function with proper coordination. RAD model distributes the analysis and construction phases into a series of short iterative development cycles. The activities of each phase per team are Business modeling, Data modeling and process modeling. This model is useful for projects with possibility of modularization. RAD may fail if modularization is difficult. This model should be used if domain experts are available with relevant business knowledge. Communication works to understand the business	***(Diagram: 2 marks Description: 2 marks, any 2 Advantages: 1 mark & any 2 Disadvantages: 1 mark)***

		problem and the information characteristics that the software must accommodate. Planning is essential because multiple software teams' work is parallel on different system functions. Modeling encompasses three major phases – business modeling, data modeling and process modeling- and establishes design representations that serve as the basis for RAD's construction activity. Construction emphasizes the use of pre-existing software components and the application of automatic code generation. Finally, deployment establishes a basis for subsequent iterations, if required. **Advantages:** 1. Changing requirements can be accommodated and progress can be measured. 2. Powerful RAD tools can reduce development time. 3. Productivity with small team in short development time and quick reviews, risk control increases reusability of components, better quality. 4. Risk of new approach only modularized systems are recommended through RAD. 5. Suitable for scalable component based systems. **Limitations/Disadvantages:** 1. RAD model success depends on strong technical team expertise and skills. 2. Highly skilled developers needed with modeling skills. 3. User involvement throughout life cycle. If developers & customers are not committed to the rapid fire activities necessary to complete the System in a much-abbreviated time frame, RAD projects will fail. 4. May not be appropriate for very large scale systems where the technical risks are high.	
	(b)	**Draw DFD level 0 and level 1 for Hotel management system.** **{**Note: Marks shall be given to any similar DFD diagrams indicating Hotel Management Activities**}**	**6M**
	Ans:	 **Level 0 DFD**  CUSTOMER Enquiry details Customer details Payment details Can. Details Room details Bill details All service details RESTAURANT Service details 0.0 Hotel Management System Rate details Tax details MIS Reports MANAGEMENT Can. details Room Details Service availed details HOUSE KEEPING **Level 1 DFD**	***(DFD Level 0: 2 marks, Level 1: 4 marks)***

	2.	**Answer any FOUR of the following :**	**16**
	(a)	**Explain process framework with suitable diagram.**	**4M**
	Ans:	A process framework establishes the foundation for a complete software process by identifying a small number of framework activities that are applicable to all software projects, regardless of their size or complexity. In addition, the process framework encompasses a set of umbrella activities that are applicable across the entire software process.  Process Framework Umbrella activities Framework activity #1 Software engineering action #1.1 Task sets Work tasks Work products Quality assurance points Project milestones Software engineering action #1.k Framework activity #n Software engineering action #n.1 Software engineering action #n.m From the above figure, each framework activity is populated by a set of software engineering actions. A collection of related tasks that produces a major software engineering work product (e.g. design is a SE action). Each action is populated with individual work tasks that accomplish some part of the work implied by the action. The following generic process framework is applicable to the vast majority of software projects. **1. Communication:** This framework activity involves heavy communication & collaboration with the customer (and the stakeholders) and encompasses requirements gathering and other related activities. **2. Planning:** This activity establishes a plan for the software engineering work that follows. It describes the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced and a work schedule. **3. Modeling:** This activity encompasses the creation of models that allow the developer & the customer to better understand software requirements & the design that will achieve those requirements. **4. Construction:** This activity combines code generation and the testing that is required uncovering errors in the code. **5. Deployment: The** software is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation. “Einstein argued that there must	***(Diagram: 2 marks, Description-: 2 marks)***

		be a simplified explanation of nature, because God is not capricious or arbitrary. No such faith comforts the software engineer. Much of the complexity that he must master is arbitrary.” **Umbrella Activities:** Generic views of SE are complemented by a set of unbrella activities. They are 1. **Software Project tracking and control:** The framework described in the generic view of SE is complemented by a number of umbrella activities, one of which is software project tracking and control. It allows the software team to access progress against the project plan and takes necessary action to maintain schedule. Umbrella activities occur throughout the software process and focus primarily on project management, tracking and control. 2. **Risk Management:** Assess risks that are likely to affect performance and quality of project. 3. **Software quality assurance:** Define and conduct activities to ensure software quality. 4. **Formal Technical Review:** Assess Software Eng. Work products to uncover and remove errors before they are shifted to next level of activity. 5. **Measurement:** Defines and collects process, project and product measures to assist the team in delivering the software that meets customer needs can be used in conjunction with all framework and umbrella activities. 6. **Software configuration Management (SCM):** Manages and effects the changes throughout the software process. 7. **Reusability management:** Defines criteria for work product reuse (including software components) and establishes the mechanism to achieve reusable components. 8. **Work product preparation and production:** Includes activities for creating work product such as models, documents, large	
	(b)	**Describe in brief four level testing processes in test execution.**	**4M**
	Ans:	Number of automated testing tools is available to the development team to reduce the cost and increase the thoroughness, accuracy of testing procedures. The four level testing done are. **Component testing:** Major part of testing time is spent for effectiveness and correctness of forms, procedure and the program. Program statements, subroutines, compound statements throughout the program are checked. **Function Testing:** This is a higher level of aggregation. Checking the performance related to components working together for the functional subsystem. This should be done after individual components are found working properly. **Subsystem Testing**: This evaluates how the related functions operate. Testing at this level checks the interrelationships of functions tested individually earlier. **System checking**: It is for checking the overall results to combine performance of all subsystems. OR	***(1 mark for each level)***

Unit Testing: During this first round of testing, the program is submitted to assessments that focus on specific units or components of the software to determine whether each one is fully functional. The main aim of this endeavor is to determine whether the application functions as designed. In this phase, a unit can refer to a function, individual program or even a procedure, and a White-box Testing method is usually used to get the job done. One of the biggest benefits of this testing phase is that it can be run every time a piece of code is changed, allowing issues to be resolved as quickly as possible. It's quite common for software developers to perform unit tests before delivering software to testers for formal testing.

Integration Testing: Integration testing allows individuals the opportunity to combine all of the units within a program and test them as a group. This testing level is designed to find interface defects between the modules/functions. This is particularly beneficial because it determines how efficiently the units are running together. Keep in mind that no matter how efficiently each unit is running, if they aren't properly integrated, it will affect the functionality of the software program. In order to run these types of tests, individuals can make use of various testing methods, but the specific method that will be used to get the job done will depend greatly on the way in which the units are defined.

System Testing: System testing is the first level in which the complete application is tested as a whole. The goal at this level is to evaluate whether the system has complied with all of the outlined requirements and to see that it meets Quality Standards. System testing is undertaken by independent testers who haven't played a role in developing the program. This testing is performed in an environment that closely mirrors production. System Testing is very important because it verifies that the application meets the technical, functional, and business requirements that were set by the customer.

Acceptance Testing: The final level, Acceptance testing (or User Acceptance Testing), is conducted to determine whether the system is ready for release. During the Software development life cycle, requirements changes can sometimes be misinterpreted in a fashion that does not meet the intended needs of the users. During this final phase, the user will test the system to find out whether the application meets their business' needs. Once this process has been completed and the software has passed, the program will then be delivered to production.

(c) **Differentiate between cardinality and modality. (Any four points)**
{Note: Any other relevant points of differentiation can also be considered}** **4M**

Ans:

(1 mark for : each difference)

Cardinality	Modality
Cardinality gives number of object occurrence in given relationship.	Modality gives existence of relationship.
Cardinality refers to the maximum number of times an instance in one entity can be associated with instance in the related entity.	Modality refers to the minimum number of times an instance in one entity can be associated with an instance in the related entity.
Cardinality can be 1 or many.	Modality can be 1 or 0
The symbol is placed on the outside ends of the relationship line, closest to the entity.	The symbol is placed on the inside, next to the cardinality symbol.
Cardinality is mandatory.	Modality is optional.

OR

Cardinality:

1. Cardinality is the specification of the number of occurrences of one [object] that can be related to the number of occurrences of another [object].
2. Cardinality is usually expressed as simply 'one' or 'many' i.e. 1:1 or 1:N or M:N
3. It also defines the max no. of objects that can participate in a relationship.
4. It does not provide an indication of whether or not a particular data object must participate in the relationship.

Modality:

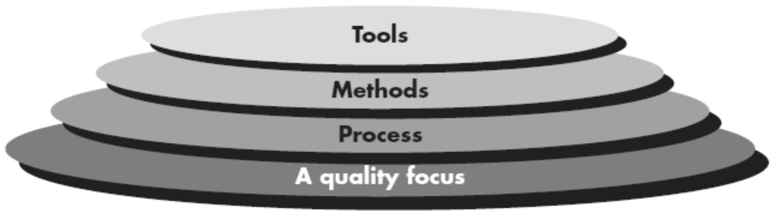
1. Cardinality does not however indicate whether or not a particular data object must participate in the relationship.
2. To specify this information, the data model adds modality to the object/relationship pair.
3. The modality of a relationship is 0 if there is no explicit need for the relationship to occur or the relationship is optional.
4. The modality is 1 if an occurrence of the relationship is mandatory.

(d) **Differentiate between waterfall and incremental model. (any four points)** **4M**

Ans:

Sr. No	Waterfall Model	Incremental Model
1	Traditional model which activities follow sequentially	Incremental model delivers series of releases called increments.
2	Users explicit prescription of requirements is essential at the start of development	Prescription of each increment is essential at the start. Additional functions and features can be added in subsequent increment
3	Time consuming model and hence customer must have patience.	Customers get quick releases with each increment and hence scope for feedback and improvement
4	Waterfall model is inappropriate for applications in which customer requirements keep changing.	Model combines the elements of waterfall model in an iterative fashion. Hence can accommodate changes during development
5	Development team size is large	Development team size is small
6	If technical errors in the development go undetected, project may be delayed or in case of major blunders results can be disastrous	Due to incremental nature, technical risks are reduced. Advantage of prompt delivery to users without hastle.
7	If any error occurred in any level it will affect outcome of all other subsequent levels.	If any error occurred can be rectify in next increment.
8	Backtracking is tedious as one has to restart project.	Backtracking can be done for uncovered errors in next increment.

(Any 4 Differences: 1 mark each)

	(e)	**Explain following with reference to design concepts in design modeling.** **(i). Abstraction** **(ii). Functional independence**	**4M**
	Ans:	**(i) Abstraction:** At the highest level of abstraction, a solution is stated in broad terms using the language of the problem environment. At lower levels of abstraction, a more detailed description of the solution is provided. As we move through different levels of abstraction, we work to create procedural and data abstractions. A procedural abstraction refers to a sequence of instructions that have a specific and limited function. An example of a procedural abstraction would be the word open for a door. A data abstraction is a named collection of data that describes a data object. In the context of the procedural abstraction open, we can define a data abstraction called door. Like any data object, the data abstraction for door would encompass a set of attributes that describe the door (e.g. door type, swing direction, weight). **(ii) Functional independence:** The concept of functional Independence is a direct outgrowth of modularity and the concepts of abstraction and information hiding. Design software such that each module addresses a specific sub-function of requirements and has a simple interface when viewed from other parts of the program structure. Functional independence is a key to good design, and to software quality. Independence is assessed using two qualitative criteria: Cohesion and Coupling. Cohesion is an indication of the relative functional strength of a module. Coupling is an indication of the relative interdependence among modules. Coupling is a qualitative indication of the degree to which a module is connected to other modules and to the outside world in "lowest possible" way.	***(2 marks Each)***
	(f)	**Explain software engineering as a layered technology approach with neat diagram.**	**4M**
	Ans:	Software engineering is a layered technology. The layers of software engineering as shown in the above diagram are:-  **1. A Quality Focus:** Any engineering approach (including software engineering) must rest on an organizational commitment to quality. Total quality management, six sigma and similar philosophies foster a continuous process improvement culture, and it is this culture that ultimately leads to the development of increasingly more effective approaches to software engineering. The bedrock that supports software engineering is a quality focus. **2. Process Layer:** The foundation for software engineering is the process layer. Software Engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software. Process defines a framework that must be established for effective delivery of software engineering technology. The software process forms the basis for management control of software projects and establishes the context in which technical methods are applied, works products (models,	***(Diagram:1 mark , Description:3 marks)***

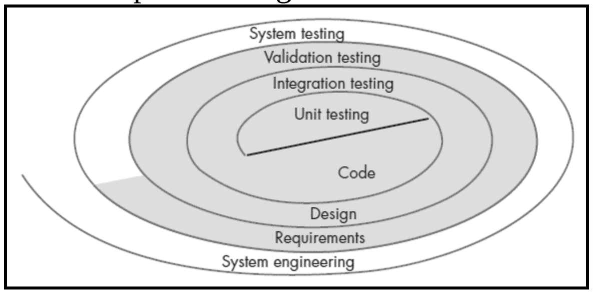
		documents, data, reports, forms etc.) are produced, milestones are established, quantity is ensured and change is properly managed. **3. Methods:** Software Engineering methods provide the technical "how to's" for building software. Methods encompass a broad array of tasks that include communication, requirements analysis, design modeling, program construction, testing and support. **4. Tools:** Software Engineering tools provide automated or semi-automated support for the process and the methods. When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer–aided software engineering is established.	
3.		**Answer any FOUR of the following:**	**16**
	(a)	**Describe Team software Process (TSP) model in detail.**	**4M**
	Ans:	Many industry-grade software projects are addressed by a team of practitioners, Watts Humphrey proposed a Team Software Process (TSP). The goal of TSP is to build a "self-directed" project team that organizes itself to produce high-quality software. Following objectives for TSP: • Build self-directed teams that plan and track their work, establish goals, and own their processes and plans. These can be pure software teams or integrated product teams (IPTs) of 3 to about 20 engineers. • Show managers how to coach and motivate their teams and how to help them sustain peak performance. • Accelerate software process improvement by making CMM Level 5 behavior normal and expected. • Provide improvement guidance to high-maturity organizations. • Facilitate university teaching of industrial-grade team skills. A self-directed team has a consistent understanding of its overall goals and objectives; defines roles and responsibilities for each team member; tracks quantitative project data (about productivity and quality); identifies a team process that is appropriate for the project and a strategy for implementing the process; defines local standards that are applicable to the team's software engineering work; continually assesses risk and reacts to it; and tracks, manages, and reports project status.TSP defines the following framework activities: project launch, high-level design, implementation, integration and test, and post-mortem. Like their counterparts in PSP (note that terminology is somewhat different), these activities enable the team to plan, design, and construct software in a disciplined manner while at the same time quantitatively measuring the process and the product. The post-mortem sets the stage for process improvements.	***(Any relevant Description: 4 marks)***
	(b)	**Describe four principles of analysis modeling.**	**4M**
	Ans:	**Principle 1: The information domain of a problem must be represented and understood.** The information domain encompasses the data that flow into the system (from end users, other systems, or external devices), the data that flow out of the system	***(Any four principles: 1 mark each)***

		(via the user interface, network interfaces, reports, graphics, and other means), and the data stores that collect and organize persistent data objects (i.e., data that are maintained permanently). **Principle 2: The functions that the software performs must be defined.** Software functions provide direct benefit to end users and also provide internal support for those features that are user visible. Some functions transform data that flow into the system. In other cases, functions effect some level of control over internal software processing or external system elements. Functions can be described at many different levels of abstraction, ranging from a general statement of purpose to a detailed description of the processing elements that must be invoked. **Principle 3: The behavior of the software (as a consequence of external events) must be represented.** The behavior of computer software is driven by its interaction with the external environment. Input provided by end users, control data provided by an external system, or monitoring data collected over a network all cause the software to behave in a specific way. **Principle 4: The models that depict information function and behavior must be partitioned in a manner that uncovers detail in a layered (or hierarchical) fashion.** Requirements' modeling is the first step in software engineering problem solving. It allows you to better understand the problem and establishes a basis for the solution (design). Complex problems are difficult to solve in their entirety. For this reason, you should use a divide-and-conquer strategy. A large, complex problem is divided into sub problems until each sub problem is relatively easy to understand. This concept is called partitioning or separation of concerns, and it is a key strategy in requirements modeling. **Principle 5: The analysis task should move from essential information toward implementation detail.** Requirements modeling begin by describing the problem from the end-user's perspective. The "essence" of the problem is described without any consideration of how a solution will be implemented. For example, a video game requires that the player "instruct" its protagonist on what direction to proceed as she moves into a dangerous maze. That is the essence of the problem.	
	(c)	**Explain general format of Software Requirement Specification (SRS).**	**4M**
	Ans:	Software requirements specification (SRS) is a document that is created when a detailed description of all aspects of the software to be built must be specified before the project is to commence. It is important to note that a formal SRS is not always written. In fact, there are many instances in which effort expended on an SRS might be better spent in other software engineering activities. However, when software is to be developed by a third party, when a lack of specification would create severe business issues, or when a system is extremely complex or business critical, an SRS may be justified. Karl Wiegers has developed a worthwhile template that can serve as a guideline for those who must create a complete SRS. A topic outline follows: 1. Introduction 1.1. Purpose	***(Any relevant Description: 4 marks)***

		1.2. Document Conventions 1.3. Intended Audience and Reading Suggestions 1.4. Project Scope 1.5. References 2. Overall Description 2.1 Product Perspective 2.2 Product Features 2.3 User Classes and Characteristics 2.4 Operating Environment 2.5 Design and Implementation Constraints 2.6 User Documentation 2.7 Assumptions and Dependencies 3. System Features 3.1 System Feature 1 3.2 System Feature 2 (and so on) 4. External Interface Requirements 4.1 User Interfaces 4.2 Hardware Interfaces 4.3 Software Interfaces 4.4 Communications Interfaces 5. Other Nonfunctional Requirements 5.1. Performance Requirements 5.2. Safety Requirements 5.3. Security Requirements 5.4. Software Quality Attributes 6. Other Requirements Appendix A: Glossary Appendix B: Analysis Models Appendix C: Issues List	
	(d)	**Describe Data Dictionary. Wire any four advantages.**	**4M**
	Ans:	Data dictionaries are lists of all of the names used in the system models. Descriptions of the entities, relationships and attributes are also included. Data dictionary as the central database for the description of all data objects. Once entries in the dictionary are defined, entity-relationship diagrams can be created and object hierarchies can be developed. Many CASE workbenches support data dictionaries. **Advantages:** 1. Support name management 2. Avoid duplication 3. Store of organizational knowledge linking analysis, design and implementation 4. It is a valuable reference in any organization because it provides documentation. 5. It improves the communication between system analyst and user by establishing consistent definitions of various items terms and procedures. 6. It is a good tool for manage operators and other members of the development team	***(Description: 2 marks any four Advantages: 2 marks)***

		to understand requirements and design. 7. It helps the analyst to simplify the structure for meeting the data requirements of the system. 8. It is just like a store of all data elements information that can link all phases of software development life cycle. 9. It is used to remove the redundancy in data definition. 10. It is an important step building a database. 11. Most data base management system has a data dictionary as a standard feature. 12. During implementation, it serves as a base against which developers compare their data description.	
	(e)	**Explain.** **(i). Components-level design elements** **(ii). Deployment level design elements.** **With respect to design model.**	**4M**
	Ans:	**Component-level design elements:** Component-level design defines data structures for all local data objects and algorithmic detail for all processing that occurs within a component and an interface that allows access to all component operations (behaviors). Within the context of object-oriented software engineering, a component is represented in UML diagrammatic form. The design details of a component can be modeled at many different levels of abstraction. A UML activity diagram can be used to represent processing logic. Detailed procedural flow for a component can be represented using either pseudo code or some other diagrammatic form. Algorithmic structure follows the rules established for structured programming. Data structures, selected based on the nature of the data objects to be processed are usually modeled using pseudo code or the programming language to be used for implementation. **Deployment-Level Design Elements:** Deployment-level design elements indicate how software functionality and subsystems will be allocated within the physical computing environment that will support the software. During design, a UML deployment diagram is developed and then refined. Each subsystem would be elaborated to indicate the components that it implements. The deployment diagram shows the computing environment but does not explicitly indicate configuration details. These details are provided when the deployment diagram is revisited in instance form during the latter stages of design or as construction begins. Each instance of the deployment (a specific, named hardware configuration) is identified.	***(Component level design elements: 2 marks Deployment level design elements: 2 marks)***
4.	**(A)**	**Answer any THREE of the following :**	**12**
	(a)	**What do you mean by testing strategies?** **{**Note: Any relevant description can also be considered**}**	**4M**
	Ans:	A strategy for software testing may also be viewed in the context of the spiral. Unit testing begins at the vortex of the spiral and concentrates on each unit (e.g., component, class, or Web App content object) of the software as implemented in source code. Testing progresses by moving outward along the spiral to integration testing, where the focus is on design and the construction of the software architecture. Taking another turn outward on the spiral, which encounters validation testing, where requirements	***(Any Four Points:1 mark Each)***

established as part of requirements modeling are validated against the software that has been constructed. Finally, system testing, where the software and other system elements are tested as a whole. To test computer software, spiral out in a clockwise direction along streamlines that broaden the scope of testing with each turn.



1. Testing begins at the component level and works "outward" toward the integration of the entire computer-based system.
2. Different testing techniques are appropriate at different points in time.
3. Testing is conducted by the developer of the software and (for large projects) an independent test group.
4. Testing and debugging are different activities, but debugging must be accommodated in any testing strategy.
5. A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high-level tests that validate major system functions against customer requirements.

(b) **Explain basic principles of project scheduling.** **4M**

Ans: *(Any Four principles: 1 mark)*

Compartmentalization: The project must be compartmentalized into a number of manageable activities, actions and tasks; both the product and the process are decomposed.
Interdependency: The interdependency of each compartmentalized activity, action or task must be determined. Some tasks must occur in sequence while others can occur in parallel. Some actions or activities cannot commence until the work product produced by another is available.
Time allocation: Each task to be scheduled must be allocated some number of work units. In addition, each task must be assigned a start date and a completion date that is a function of the interdependencies.
Start and stop dates are also established based on whether work will be conducted on a full-time or part-time basis.
Effort validation: Every project has a defined number of people on the team. As time allocation occurs, the project manager must ensure that no more than the allocated number of people has been scheduled at any given time.
Defined responsibilities: Every task that is scheduled should be assigned to a specific team member.
Defined outcomes: Every task that is scheduled should have a defined outcome for software projects such as a work product or part of a work product. Work products are often combined in deliverables.
Defined milestones: Every task or group of tasks should be associated with a project milestone. A milestone is accomplished when one or more work products has been reviewed for quality and has been approved.

	(c)	**Write steps to perform statistical SQA.**	**4M**
	Ans:	Statistical quality assurance reflects a growing trend throughout industry to become more quantitative about quality. For software, statistical quality assurance implies the following steps: 1. Information about software defects is collected and categorized. 2. An attempt is made to trace each defect to its underlying cause (e.g., nonconformance to specifications, design error, violation of standards, and poor communication with the customer) 3. Using the Pareto principle (80 percent of the defects can be traced to 20 percent of all possible causes), isolate the 20 percent (the "vital few"). 4. Once the vital few causes have been identified, move to correct the problems that have caused the defects.	***(For each point: 1 mark)***
	(d)	**Explain smoke testing with its advantages and disadvantages (2 each)**	**4M**
	Ans:	Smoke testing is an integration testing approach that is commonly used when ―shrink wrapped software products are being developed. It is designed as a pacing mechanism for time-critical projects, allowing the software team to assess its project on a frequent basis. In essence, the smoke testing approach encompasses the following activities: Software components that have been translated into code are integrated into a ―build. A build includes all data files, libraries, reusable modules, and engineered components that are required to implement one or more product functions. A series of tests is designed to expose errors that will keep the build from properly performing its function. The intent should be to uncover show stopper errors that have the highest likelihood of throwing the software project behind schedule. The build is integrated with other builds and the entire product (in its current form) is smoke tested daily. The integration approach may be top down or bottom up. The daily frequency of testing the entire product may surprise some readers. However, frequent tests give both managers and practitioners a realistic assessment of integration testing progress. McConnell describes the smoke test in the following manner: The smoke test should exercise the entire system from end to end. It does not have to be exhaustive, but it should be capable of exposing major problems. The smoke test should be thorough enough that if the build passes, you can assume that it is stable enough to be tested more thoroughly. Smoke testing provides a number of benefits when it is applied on complex, time critical software engineering projects. **Advantages:** • Integration risk is minimized. • Error diagnosis and correction are simplified. • Progress is easier to assess. **Disadvantages:** • The smoke test should exercise the entire system from end to end. • The smoke test should be thorough enough. • A dedicated team is requiring performing testing on individual build.	***(Description: 2 marks Advantages: 1 mark Disadvantages: 1 mark)***
	(B)	**Answer any ONE of the following:**	**6**
	(a)	**Explain CPM. How is it different from PERT?**	**6M**
	Ans:	**Critical Path Method (CPM):** CPM is a technique that is used in projects that have predictable activities and tasks such as in construction projects.	***(CPM: 2 marks,***

- It allows project planners to decide which aspect of the project to reduce or increase when a trade-off is needed.
- It is a deterministic tool and provides an estimate on the cost and the amount of time to spend in order to complete the project.
- It allows planners to control both the time and cost of the project.

Difference: 4 marks any 4 points)

CPM differs from PERT as follow:

- The Program Evaluation and Review Technique (PERT) is a project management technique or tool which is suitable for projects that have unpredictable activities while the Critical Path Method (CPM) is a project management tool which is suitable for projects that have predictable activities.
- CPM uses a single estimate for the time that a project can be completed while PERT uses three estimates for the time that it can be completed.
- CPM is a deterministic project management tool while PERT is a probabilistic project management tool.
- CPM allows project management planners to determine which aspect of the project to sacrifice when a trade-off is needed in order to complete the project while PERT does not.

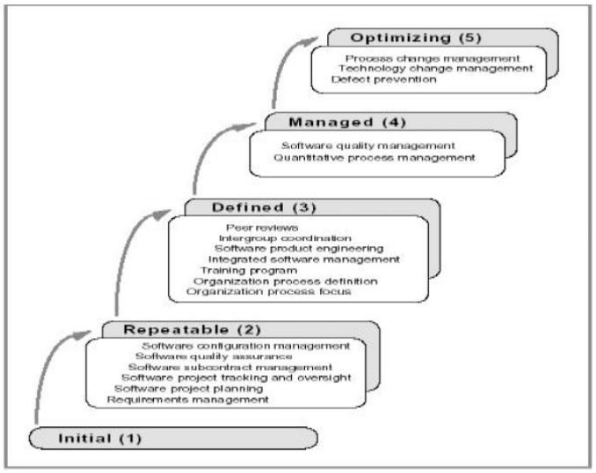
OR

PERT	CPM
The Program Evaluation and Review Technique (PERT) is a project management technique or tool which is suitable for projects that have unpredictable activities.	Critical Path Method (CPM) is a project management tool which is suitable for projects that have predictable activities.
PERT uses three estimates for the time that it can be completed.	CPM uses a single estimate for the time that a project can be completed.
PERT is a probabilistic project management tool.	CPM is a deterministic project management tool.
PERT does not allows project management planners to determine which aspect of the project to sacrifice when a trade-off is needed in order to complete the project.	CPM allows project management planners to determine which aspect of the project to sacrifice when a trade-off is needed in order to complete the project.
PERT tool is basically a tool for planning	CPM also allows an explicit estimate of control of time, costs in addition to time, therefore CPM can control both time and cost.
PERT is more suitable for R&D related	CPM is best suited for routine and those projects where the project is performed for projects where time and cost estimates can the first time and the estimate of duration be accurately calculated are uncertain.
PERT uses event oriented Network.	CPM uses activity oriented network.

	(b)	**Explain CMMI with its levels and neat diagram.**	**6M**

Ans:

Definition: Capability Maturity Model Integration (CMMI) is a process improvement approach that helps organizations improves their performance.


Optimizing (5)
Process change management
Technology change management
Defect prevention
Managed (4)
Software quality management
Quantitative process management
Defined (3)
Peer reviews
Intergroup coordination
Software product engineering
Integrated software management
Training program
Organization process definition
Organization process focus
Repeatable (2)
Software configuration management
Software quality assurance
Software subcontract management
Software project tracking and oversight
Software project planning
Requirements management
Initial (1)

CMMI Levels:

Level 1: Initial. The software process is characterized as ad hoc and occasionally even chaotic. Few processes are defined, and success depends on individual effort.

Level 2: Repeatable. Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

Level 3: Defined. The software process for both management and engineering activities is documented, standardized, and integrated into an organization wide software process. All projects use a documented and approved version of the organization's process for developing and supporting software. This level includes all characteristics defined for level

Level 4: Managed. Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled using detailed measures. This level includes all characteristics defined for level

Level 5: Optimizing. Continuous process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technologies. This level includes all characteristics defined for level 4.

OR

Level 0: Incomplete—the process area (e.g., requirements management) is either not performed or does not achieve all goals and objectives defined by the CMMI for level 1 capability for the process area.

Level 1: Performed—all of the specific goals of the process area (as defined by the CMMI) have been satisfied. Work tasks required to produce defined work products are being conducted.

Level 2: Managed—all capability level 1 criteria have been satisfied. In addition, all work associated with the process area conforms to an organizationally defined policy; all people doing the work have access to adequate resources to get the job done; stakeholders are actively involved in the process area as required; all work tasks and work products are "monitored, controlled, and reviewed; and are evaluated for adherence to the process description".

(Description with level: 4 marks & diagram: 2 marks)

		Level 3: Defined—all capability level 2 criteria have been achieved. In addition, the process is "tailored from the organization's set of standard processes according to the organization's tailoring guidelines, and contributes work products, measures, and other process-improvement information to the organizational process assets". **Level 4: Quantitatively managed**—all capability level 3 criteria have been achieved. In addition, the process area is controlled and improved using measurement and quantitative assessment. "Quantitative objectives for quality and process performance are established and used as criteria in managing the process". **Level 5: Optimized**—all capability level 4 criteria have been achieved. In addition, the process area is adapted and optimized using quantitative (statistical) means to meet changing customer needs and to continually improve the efficacy of the process area under consideration.	
5.		**Answer any TWO of the following :**	**16**
	(a)	**State and describe various core principles of software engineering.**	**8M**
	Ans:	**The First Principle: The Reason It All Exists A** software system exists for one reason: To provide value to its users. All decisions should be made with this in mind. Before specifying a system requirement, before noting a piece of system functionality, before determining the hardware platforms or development processes, ask yourself questions such as: "Does this add real VALUE to the system?" If the answer is "no", don't do it. All other principles support this one. **The Second Principle: Keep it Simple and Stupid** Software design is not haphazard process. There are many factors to consider in any design efforts. All design should be as simple as possible, but no simpler. This facilitates having a more easily understood, and easily maintained system. This is not to say that features, even internal features, should be discarded in the name of simplicity. **The Third Principle: Maintain the Vision** A Clear vision is essential to the success of a software project. Without that A project almost unfailingly ends up being —of two minds about itself. Without conceptual integrity, a system threatens to become a patchwork of incompatible designs, held together. **The Fourth Principle: What You Produce, Others Will Consume** Seldom is an industrial strength software system constructed and used in vacuum. In some way or other, someone else will use, maintain, document or otherwise depend on being able to understand your system. So always specify design and implement knowing someone else will have to understand what you are doing. **The Fifth Principle: Be Open to the Future** While implementing a system a developer of a system shall always consider for the scope of improvement in a given system, by virtue of which there will be continuous upgrading in a given system and the system can be timely reviewed. As a result of this one can have better and improved system at the end of all iterations. **The Six Principle: Plan Ahead For Reuse** While implementing a system the development team can think about having a module which are already developed in a similar kind of system, resulting in a less time of development and as these modules are reused very little time can be spend on the	***(List of all principles: 1 mark; Description of Each principle: 1 mark)***

		testing of such module. The fact is few testing strategies like unit testing can be avoided as more often than not such modules are already working in a desired manner. **The Seven Principle: Think** During the implementation phase; one shall think about the implementation of the system. More you think one a system better system can be implemented as better and improved ideas can be evolve. By thinking on a system in broader way it give a broader scope of a system.	
	(b)	**Describe the following with respect to Risk assessment :** **(i). Risk identification** **(ii). Risk analysis** **(iii). Risk prioritization**	**8M**
	Ans:	**Risk identification**: Is the first step in risk assessment, which identifies all the different risks for a particular project. These risks are project-dependent and identifying them is an exercise in envisioning what can go wrong. Methods that can aid risk identification include checklists of possible risks, surveys, meetings and brainstorming, and reviews of plans, processes, and work products. Checklists of frequently occurring risks are probably the most common tool for risk identification—most organizations prepare a list of commonly occurring risks for projects, prepared from a survey of previous projects. Such a list can form the starting point for identifying risks for the current project. Risk identification merely identifies the undesirable events that might take place during the project, i.e., enumerates the “unforeseen” events that might occur. It does not specify the probabilities of these risks materializing nor the impact on the project if the risks indeed materialize. Hence, the next tasks are risk analysis and prioritization **Risk analysis:** In **risk analysis**, the probability of occurrence of a risk has to be estimated, along with the loss that will occur if the risk does materialize. This is often done through discussion, using experience and understanding of the situation, though structured approaches also exist. Once the probabilities of risks materializing and losses due to materialization of different risks have been analyzed, they can be **prioritized**. **Risk prioritization:** One approach for prioritization is through the concept of risk exposure, which is sometimes called risk impact. RE is defined by the relationship RE = Prob(UO) * Loss(UO), Where Prob(UO) is the probability of the risk materializing (i.e., undesirable outcome) and Loss(UO) is the total loss incurred due to the unsatisfactory outcome. The loss is not only the direct financial loss that might be incurred but also any loss in terms of credibility, future business, and loss of property or life. The RE is the expected value of the loss due to a particular risk. For risk prioritization using RE is, the higher the RE, the higher the priority of the risk item.	***(Risk identification: 3 marks, Risk analysis: 3 marks, Risk prioritization :2 marks)***
	(c)	**What is Quality Assurance? Describe various SQA activities.**	**8M**
	Ans:	Software quality assurance is composed of a variety of tasks associated with two different constituencies - the software engineers who do technical work and an SQA group that has responsibility for quality assurance planning, oversight, record keeping, analysis, and reporting. Software engineers address quality (and perform quality	***(Explanation of SQA: 2 marks; six***

		assurance and quality control activities) by applying solid technical methods and measures, conducting formal technical reviews, and performing well-planned software testing. **Activities of SQA:** 1) **Prepare an SQA plan for a project:** The plan is developed during project planning and is reviewed by all interested parties. Quality assurance activities performed by the software engineering team and the SQA group are governed by the plan. The plan identifies • evaluations to be performed • audits and reviews to be performed • standards that are applicable to the project • procedures for error reporting and tracking • documents to be produced by the SQA group • amount of feedback provided to the software project team 2) **Participate in the development of the project's software process description**: The software team selects a process for the work to be performed. The SQA group reviews the process description for compliance with organizational policy, internal software standards, externally imposed standards (e.g., ISO-9001), and other parts of the software project plan. 3) **Review software engineering activities to verify compliance with the defined software process:** The SQA group identifies, documents, and tracks deviations from the process and verifies that corrections have been made. 4) **Audits designated software work products to verify compliance with those defined as part of the software process:** The SQA group reviews selected work products; identifies, documents, and tracks deviations; verifies that corrections have been made; and periodically reports the results of its work to the project manager. 5) **Ensure that deviations in software work and work products are documented and handled according to a documented procedure:** Deviations may be encountered in the project plan, process description, applicable standards, or technical work products. 6) **Records any noncompliance and reports to senior management:** Noncompliance items are tracked until they are resolved.	***Activities of SQA: 6 marks)***
6.		**Answer any FOUR of the following:**	**16**
	(a)	**Differentiate between verification and validation.**	**4M**
	Ans:	**1. Verification:** **1)** Verification is a set of activities which ensures that software correctly implements a specific function. **2)** Verification evaluates plans, documents, code, requirements and specifications etc. **3)** Verification takes place before validation. **4)** The input of verification could be check list, review of meetings, plans, walkthroughs, etc. **2. Validation:** **1)** Validation is different set of activities which ensures that the software has been built is traceable to customer.	***(Any 4 differences: 4 marks, per difference 1 mark)***

2) Validation evaluates product itself.
3) Validation is next step to verification.
4) An input of validation is actual testing of product

OR

Verification	Validation
It answers the questions like: Am I building the product right?	It answers the question like: Am I building the right product?
Verification is a static practice of verifying documents, design, code and program.	Validation is a dynamic mechanism of validating and testing the actual product.
It does not involve executing the code.	It always involves executing the code.
It is human based checking of documents and files.	It is computer based execution of program
Verification uses methods like inspections, reviews, walkthroughs, and Desk-checking etc	Validation uses methods like black box (functional) testing, gray box testing, and white box (structural) testing etc.
Verification is to check whether the software conforms to specifications	Validation is to check whether software meets the customer expectations and requirements.
It can catch errors that validation cannot catch. It is low level exercise.	It can catch errors that verification cannot catch. It is High Level Exercise.
Target is requirements specification, application and software architecture, high level, complete design, and database design etc.	Target is actual product-a unit, a module, a bent of integrated modules, and effective final product.
It generally comes before validation.	It generally follows after verification.

(b) **Explain Brute force approaches used in debugging strategies.**
{Note: Any relevant description can be considered**}** **4M**

Ans: 1. **Brute Force:** This category of debugging is probably the most common and least efficient method for isolating the cause of a software error. Brute force debugging methods are applied when all else fails. Using a "let the computer find the error" philosophy, memory dumps are taken, run-time traces are invoked, and the program is loaded with WRITE statements. In the morass of information that is produced a clue is found that can lead us to the cause of an error. Although the mass of information produced may ultimately lead to success, it more frequently leads to wasted effort and time. Thought must be expended first.

(Explanation: 4 marks)

(c) **Explain any four features of SCM.** **4M**

Ans: **SCM Features:** To support SCM, the repository must have a tool set that provides support for the following features:

(4 features: 4 marks, 1

		Versioning: As a project progresses, many versions of individual work products will be created. The repository must be able to save all of these versions to enable effective management of product releases and to permit developers to go back to previous versions during testing and debugging. The repository must be able to control a wide variety of object types, including text, graphics, bit maps, complex documents, and unique objects like screen and report definitions, object files, test data, and results. **Dependency tracking and change management:** The repository manages a wide variety of relationships among the data elements stored in it. These include relationships between enterprise entities and processes, among the parts of an application design, between design components and the enterprise information architecture, between design elements and deliverables, and so on. Some of these relationships are merely associations, and some are dependencies or mandatory relationships. **Requirements tracing:** This special function depends on link management and provides the ability to track all the design and construction components and deliverables that result from a specific requirements specification (forward tracing). In addition, it provides the ability to identify which requirement generated any given work product (backward tracing). **Configuration management:** A configuration management facility keeps track of a series of configurations representing specific project milestones or production releases. **Audit trails:** An audit trail establishes additional information about when, why, and by whom changes are made. Information about the source of changes can be entered as attributes of specific objects in the repository. A repository trigger mechanism is helpful for prompting the developer or the tool that is being used to initiate entry of audit information (such as the reason for a change) whenever a design element is modified.	***mark per feature)***
	(d)	**Explain the following management spectrum:** **(i). The Process** **(ii). The Project**	**4M**
	Ans:	**Software Process:** A software process provides the framework from which a comprehensive plan for software development can be established. A number of different tasks sets-tasks; milestones, work products, and quality assurance points enable the framework activities to be adapted to the characteristics of the software projects and the requirements of the project. **Software Product:** Before a project can be planned, product objectives and scope should be established, alternative solutions should be considered, and technical and management constrains should be identified. Without this information, it is impossible to define reasonable and accurate estimates of the cost, an effective assignment of risk, a realistic breakdown of project tasks, or a manageable product schedule that provides a meaningful indication of progress. The developers must examine the product and the problem intended to solve at the very beginning of the project. For this reason, the scope of the product must be established and bounded.	***(2 marks each)***
	(e)	**Describe white box and black box testing of software.**	**4M**
	Ans:	**White Box Testing:** White-box tests, sometimes called glass-box testing or code testing, is a test case design method that uses the control structure of the procedural	***(Description of white box***

design to derive test cases. Using white-box testing methods, the software engineer can derive test cases that:

1. Guarantee that all independent paths within a module have been exercised at least once.
2. Exercise all logical decisions on their true and false sides,
3. Execute all loops at their boundaries and within their operational bounds, and
4. Exercise internal data structures to ensure their validity.

Even though white box testing seems to be an ideal method for testing, it does not guarantee failures from faulty data. Secondly it is difficult to conduct such extensive and exhaustive testing in large organizations.

and black box: 2 marks each)

OR

White Box Testing (also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing or Structural Testing) is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester. The tester chooses inputs to exercise paths through the code and determines the appropriate outputs. Programming know-how and the implementation knowledge is essential. White box testing is testing beyond the user interface and into the nitty-gritty of a system. This method is named so because the software program, in the eyes of the tester, is like a white/ transparent box; inside which one clearly sees.

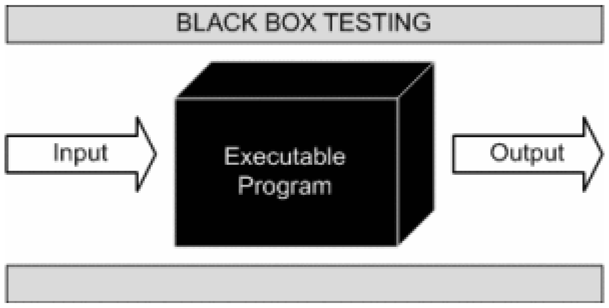
EXAMPLE: A tester, usually a developer as well, studies the implementation code of a certain field on a webpage, determines all legal (valid and invalid) AND illegal inputs and verifies the outputs against the expected outcomes, which is also determined by studying the implementation code. White Box Testing is like the work of a mechanic who examines the engine to see why the car is not moving.

Black -box testing: Black Box testing: In this method the analyst examines the specifications to see what the program must do under various conditions. Test cases are then developed for a condition or a combination of conditions and submitted for processing. By examining the results the analyst can determine if the specified requirements are satisfied. Black-box testing attempts to find errors in the following categories:

OR

Black box testing:

Black Box Testing, also known as Behavioural Testing, is a software testing method in which the internal structure/ design/ implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional.



		This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see. This method attempts to find errors in the following categories: • Incorrect or missing functions • Interface errors • Errors in data structures or external database access • Behaviour or performance errors • Initialization and termination errors **EXAMPLE:** • A tester, without knowledge of the internal structures of a website, tests the web pages by using a browser; providing inputs (clicks, keystrokes) and verifying the outputs against the expected outcome.	