MODEL ANSWER

SUMMER - 2017 EXAMINATION

Subject: Software Engineering **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 tryto assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling errors should not be given moreImportance (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)** Ans.	**Attempt any FIVE of the following:** **What is software? What is embedded software?** Software is (1) Instructions (computer programs) that when executed provide desired function and performance, (2) Data structures that enable the programs to adequately manipulate information, and (3) Documents that describe the operation and use of the programs **Embedded software:** Intelligent products have become commonplace in nearly every consumer and industrial market. Embedded software resides in read-only memory and is used to control products and systems for the consumer and industrial markets. Embedded software can perform very limited and esoteric functions (e.g., keypad 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, and braking	**20** **4M** ***Definition or concept of software 2M*** ***Embedded software 2M***

		systems). A function point extension called feature points. It is a superset of the function point measure that can be applied to systems and engineering software applications. The feature point measure accommodates applications in which algorithmic complexity is high. Real-time, process control and embedded software applications tend to have high algorithmic complexity and are therefore amenable to the feature point.	
	b) Ans.	**Explain the term scrum.** Scrum is a management and controls process that cuts through complexity to focus on building software that meets business needs. Scrum itself is a framework for effective team collaboration on complex software projects. They need real time and fast decision making process which is need for getting accurate information and on actual events. Teams in the organization work together in a team focusing on the organizational goals that they need to achieve. The team is hard working and goal oriented even though it is a small team work. Scrum is a repetitive and incremental framework for project management majorly used in very active software development. Scrum methodology gives premium to functional software, the freedom to change along with new business realities, collaboration and communication. It is a flexible, holistic strategy of product development in which a team of developers works as a unit in order to accomplish an objective that is common to them all, challenging assumptions of the “traditional, sequential approach” to product development. The scrum methodology requires openness and trust in the team, which these five values of Scrum support • Openness: Members of the team and their stakeholders consent to be open about their work and any issues they encounter. • Commitment: Members of the team individually promise to reach their team goals, in every Sprint. • Courage: Team members know they are courageous to work through disagreement and issues together so they can do what is right. • Respect: There is respect among team members to be capable technically as well as to work with the right intent. • Focus: Members of the Team concentrate exclusively on the goals	**4M** ***Definition or concept of scrum 2M*** ***Explanation 2M***

		of their team and the Sprint Backlog; no work should be done outside their backlog.	
	c) Ans.	**List core principle of Software Engineering.** **Following are the core principles of software engineering** 1. TheReasonIt AllExists 2. KeepIt Simple,Stupid! 3. Maintain theVision 4. WhatYou Produce, Others WillConsume. 5. BeOpento theFuture 6. PlanAhead forReuse 7. Think! **1. 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. **2. Keep It Simple, Stupid!** There are many factors to consider in any design effort. All design should be as simple as possible, but no simpler. This facilitates having a more easily understood, and easily maintained system. **3. Maintain the Vision** A clear vision is essential to the success of a software project. Without one, a project almost unfailingly ends up being "of two [or more] minds" about itself. Compromising the architectural vision of a software system weakens and will eventually break even the most well designed systems. Having an empowered Architect who can hold the vision and enforce compliance helps ensure a very successful software project. **4. What You Produce, Others Will Consume** Seldom is an industrial-strength software system constructed and used in a vacuum. In some way or other, someone else will use, maintain, document, or otherwise depend on being able to	**4M** ***List of correct principle explanat ion2M*** ***2M forAny 1 or 2 expected to explain in short***

understand your system. So, always specify, design, and implement knowing someone else will have to understand what you are doing. The audience for any product of software development is potentially large. Making their job easier adds value to the system.

5. Be Open to the Future

A system with a long lifetime has more value. In today's computing environments, where specifications change on a moment's notice and hardware platforms are obsolete when just a few months old, software lifetimes are typically measured in months instead of years. However, true "industrial-strength" software systems must endure far longer. To do this successfully, these systems must be ready to adapt to these and other changes. Systems that do this successfully are those that have been designed this way from the start. Never design yourself into a corner. Always ask "what if ", and prepare for all possible answers by creating systems that solve the general problem, not just the specific one. This could very possibly lead to the reuse of an entire system.

6. Plan Ahead for Reuse

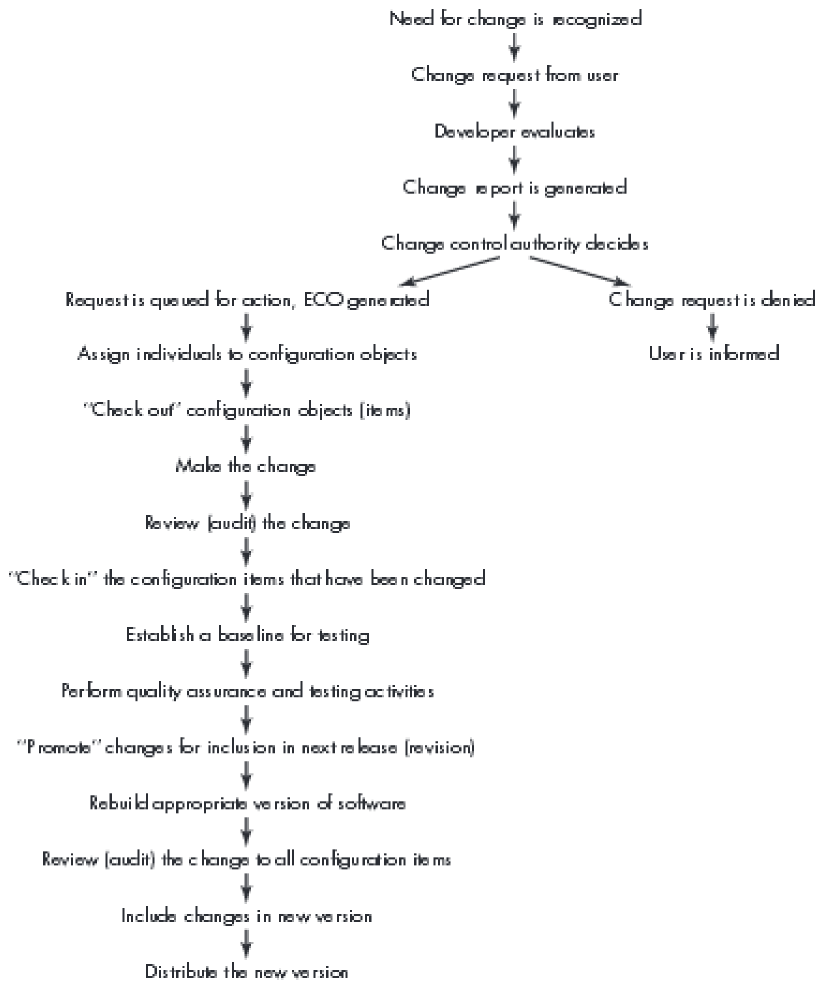
Reuse saves time and effort. Achieving a high level of reuse is arguably the hardest goal to accomplish in developing a software system. The reuse of code and designs has been proclaimed as a major benefit of using object-oriented technologies. However, the return on this investment is not automatic. How can you reuse something that you don't know exists? Planning ahead for reuse reduces the cost and increases the value of both the reusable components and the systems into which they are incorporated.

7. Think!

This last Principle is probably the most overlooked. Placing clear, complete thought before action almost always produces better results. When you think about something, you are more likely to do it right. You also gain knowledge about how to do it right again. If you do think about something and still do it wrong, it becomes valuable experience. A side effect of thinking is learning to recognize when you don t know something, at which point you can research the answer. When clear thought has gone into a system, value comes out. Applying the first six Principles requires intense thought, for which the potential rewards are enormous.

	d) Ans.	**Write importance of analysis modelling.** ***(Note: Four relevant importanceshallbeconsider)*** **Importance of analysis modeling:** 1.Designing gets easier to the designer 2.Better understanding of system can be accomplished 3.System feasibility can be determined 4.Defines data objects 5.Describes data attributes 6.Establishes data relationships 7.Identifies functions that transform data objects 8.Indicates different states of the system 9.Specifies events that cause the system to change state 10.Refines each model to represent lower levels of abstraction 11.Refines data objects 12. Relates a functional hierarchy represent behavior at different levels of detail	**4M** ***For each importance 1M***
	e) Ans.	**List various testing characteristics.** Objective of testing is to find errors and good test must have high probability of finding an error. • **Testability:** How software can be tested? It should not require more time to test. To reduce testing cost and time • **Operability:** It should be easily operable by end user to get desired output. System should be designed and implemented keeping quality in mind. System should be more user friendly. • **Observability:** What you can see that you can test. Inputs are provided as a part of test during testing execution. If system states and variables are visible incorrect output can be easily identified. Hence internal errors can be detected and reported. • Controllability: Better we control software, the most testing can be automated and optimized. • **Decomposability:** The program should be decomposable into small modules as a result a small testing module can be implemented to check validity. The software system is built from no. of independent modules that can be tested independently. • **Simplicity:** Testing can be done quickly if there is only less to test. The software program should exhibit functionally, structurally, code based simple. • **Stability:** Uncontrolled and frequent changes cause much more disturbance to the test activity. Changes should be as few as	**4M** ***List of testing characteristics 2M and any four to six point with explanation 2M***

		possible. The software should be able to recover well from changes or failures. • **Understandability:** If one can understand the software more then we can test software smartly. The internal, external interfaces and dependencies between system components should be clearly understood. Any change made to design and implement should be informed to tester.	
	f) Ans.	**What is change control?** **Change Control:** Change control is vital. But the forces that make it necessary also make it annoying. We worry about change because a tiny perturbation in the code can create a big failure in the product. But it can also fix a big failure or enable wonderful new capabilities. We worry about change because a single rogue developer could sink the project; yet brilliant ideas originate in the minds of those rogues, and a burdensome change control process could effectively discourage them from doing creative work. For a large software project, uncontrolled change rapidly leads to chaos. For such projects, change control combines human procedures and automated tools to provide a mechanism for the control of change. A change request is submitted and evaluated to assess technical merit, potential side effects, overall impact on other configuration objects and system functions, and the projected cost of the change. The results of the evaluation are presented as a change report, which is used by a change control authority (CCA)—a person or group that makes a final decision on the status and priority of the change. An engineering change order (ECO) is generated for each approved change. The ECO describes the change to be made, the constraints that must be respected, and the criteria for review and audit. The object(s) to be changed can be placed in a directory that is controlled solely by the software engineer making the change. A version control system updates the original file once the change has been made. As an alternative the object(s) to be changed can be "checked out" of the project database (repository), the change is made, and appropriate SQA activities are applied. The object(s) is (are) then "checked in" to the database and appropriate version control mechanisms are used to create the next version of the	**4M** ***Explanation relevant to change control 2M***

		software. These version control mechanisms, integrated within the change control process, implement two important elements of change management—access control and synchronization control. Access control governs which software engineers have the authority to access and modify a particular configuration object. Synchronization control helps to ensure that parallel changes, performed by two different people, don't overwrite one another. **The Change Control Process**  Need for change is recognized Change request from user Developer evaluates Change report is generated Change control authority decides Request is queued for action, ECO generated Change request is denied User is informed Assign individuals to configuration objects "Check out" configuration objects (items) Make the change Review (audit) the change "Check in" the configuration items that have been changed Establish a baseline for testing Perform quality assurance and testing activities "Promote" changes for inclusion in next release (revision) Rebuild appropriate version of software Review (audit) the change to all configuration items Include changes in new version Distribute the new version	***Diagram 2M***

	g) Ans.	**List various activities of SQA.** 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 assurance and quality control activities) by applying solid technical methods and measures, conducting formal technical reviews, and performing well-planned software testing. **SQA Activities: (Any 4-6 activities accepted)** These activities are performed (or facilitated) by an independent SQA group that: 1. Evaluations to be performed 2. Prepares an SQA plan for a project. 3. Participates in the development of the project's software process description. 4. Reviews software engineering activities to verify compliance with the defined software process. 5. Audits designated software work products to verify compliance with those defined as part of the software process. 6. Ensures that deviations in software work and work products are documented and handled according to a documented procedure. 7. Records any noncompliance and reports to senior management. 8. Audits and reviews to be performed 9. Standards those are applicable to the project 10. Procedures for error reporting and tracking 11. documents to be produced by the SQA group 12. Amount of feedback provided to the software project team	**4M** ***Description of SQA - 1M*** ***Activities of SQA- 3M***
2.	 **a)** Ans.	**Attempt any <u>FOUR</u> of the following:** **What is quality control?** Quality Control is known as QC and focuses on identifying defect. QC ensures that the approaches, techniques, methods and processes are designed in the project are following correctly. QC activities monitor and verify that the project deliverables meet the defined quality standards. Quality Control is a reactive process and is detection in nature. It recognizes the defects. Quality Control has to complete after Quality	**16** **4M** ***Concept 2M***

Assurance.
The basic definition of quality for a software product is the product's ability to fulfill the user's reasonable wishes and expectations, it must provide functions of a type and at a time when the user needs them, and the product must work. A closer inspection of what perceived quality consists of reveals that there are many dimensions to defining a quality product.
Some of these dimensions are:

- The level of satisfaction that the user perceives from the product.
- Achieving a perceived high quality with this demands that the product also meets the customer's implied needs of the product.
- The value of the product. This value is perceived for all stakeholders relative to their competitive environment.
- Key attributes, such as reliability, usability, efficiency, portability, maintainability, Defectiveness, i.e. the degree to which the product works incorrectly

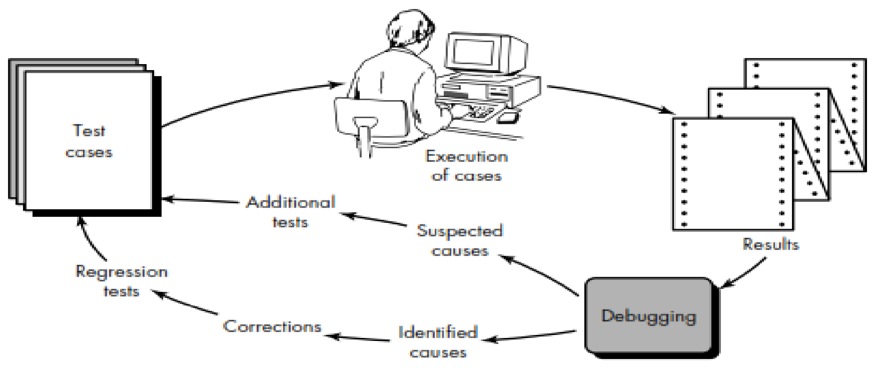
Software Quality Control (SQC) is a set of activities for ensuring quality in software products.
It includes the following activities:

- ***Reviews***
  - Requirement Review
  - Design Review
  - Code Review
  - Deployment Plan Review
  - Test Plan Review
  - Test Cases Review
- ***Testing***
  - Unit Testing
  - Integration Testing
  - System Testing
  - Acceptance Testing

2M for importance of Quality control

Software Quality Control is limited to the Review/Testing phases of the Software Development Life Cycle and the goal is to ensure that the products meet specifications/requirements.
The process of Software Quality Control (SQC) is governed by Software Quality Assurance (SQA). While SQA is oriented towards prevention, SQC is oriented towards detection.

	b) Ans.	**Explain following terms w.r.t. risk management:** **(i) Risk identification** **(ii) Risk analysis** **(i) Risk identification:** During the first step in the software risk management process, risks are identified and added to the list of known risks. The output of this step is a list of project-specific risks that have the potential of compromising the project's success. There are many techniques for identifying risks, including interviewing, reporting, decomposition, assumption analysis, critical path analysis, and utilization of risk taxonomies. Interviewing/Brainstorming: One technique for identifying risks is interviewing or brainstorming with project personnel, customers, and vendors. Open-ended questions such as the following can help identify potential areas of risk. What new or improved technologies does this project implement? What interfaces issues still need to be defined? What requirements exist that we aren't sure how to implement? What concerns do we have about our ability to meet the required quality and performance levels? Voluntary Reporting: Decomposition: As the product is being decomposed during the requirements and design phases, another opportunity exists for risk identifications. Every TBD ("To Be Done/Determined") is a potential risk. "The most important thing about planning is writing down what you don't know, because what you don't know is what you must find out. Decomposition in the form of work breakdown structures during project planning can also help identify areas of uncertainty that may need to be recorded as risks. Assumption Analysis: Process and product assumptions must be analyzed. Risk Taxonomies: Risk taxonomies are lists of problems that have occurred on other projects and can be used as checklists to help ensure all potential risks have been considered. **(ii) Risk analysis** During the risk analysis step, each risk is assessed to determine: Likelihood: the probability that the risk will result in a loss Impact: the size or cost of that loss if the risk turns into a problem	**4M** ***2M for Risk identific ation*** ***2M for Risk***

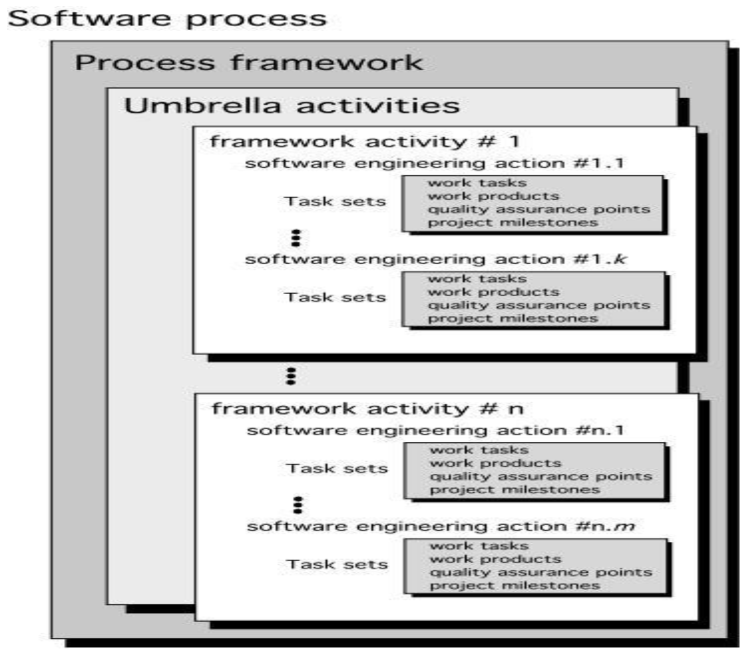
		Timeframe: when the risk needs to be addressed (i.e., risk associated with activities in the near future would have a higher priority similar risks in later activities), the interrelationships between risks are assessed to determine if compounding risk conditions magnify losses. Comparing the Risk Exposure measurement for various risks can help identify those risks with the greatest probable negative impact to the project or product and thus help establish which risks are candidates for further action. The list of risks is then prioritized based on the results of our risk analysis. Since resource limitations rarely allow the consideration of all risks, the prioritized list of risks is used to identify risks requiring additional planning and action. Other risks are documented and tracked for possible future consideration. Risk analysis is based on changing conditions, additional information, the identification of new risks, or the closure of existing risks, the list of risks requiring additional overheads.	***Analysis***
	c) Ans.	**Describe debugging process.**  Test cases Execution of cases Results Debugging Suspected causes Additional tests Regression tests Corrections Identified causes **The Debugging Process** The execution of cases suspected causes, Identified causes, is Debugging. Debugging is not testing but always occurs as a consequence of testing. The debugging process begins with the execution of a test case. Results are assessed and a lack of correspondence between expected and actual performance is encountered. In many cases, the non-corresponding data are a symptom of an underlying cause as yet hidden. The debugging process attempts to match symptom with cause, thereby leading to error correction.	**4M** ***Definition or concept 2M*** ***2M for importance of***

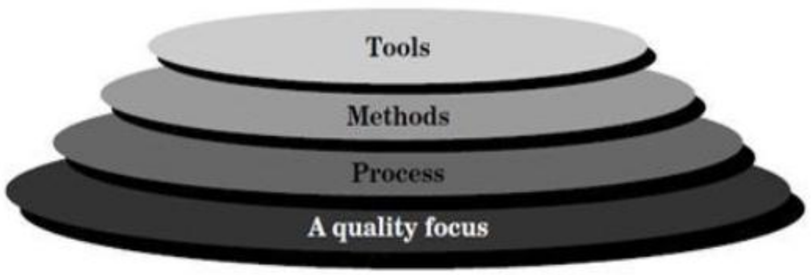
		The debugging process will always have one of two outcomes: (1) The cause will be found and corrected, or (2) The cause will not be found. In the latter case, the person performing debugging may suspect a cause, design a test case to help validate that suspicion, and work toward error correction in an iterative fashion. Why is debugging so difficult? Human psychology has more to do with an answer than software technology.	***Quality control***
	d) Ans.	**Compare cardinality and Modality.** ***(AnyFour relevant Points shall be considered)***	**4M** ***Any four points 1M each***

Cardinality	Modality
Cardinality defines the rangeof object-to object Relationships	Modalityindicates whether ornot a relationship between objects is mandatory
Expected Values are1:1, 1:N, N:M	Expected values are0 or1 only
It does not, however, provide an indication of whetheror notaparticular data object must participate in the relationship.	It provides indication ofparticipation in the relationship byhavingvalues as 1, if value is 0 no participation in relationship will exist.
Itgives maximum numbers occurrences inrelationship	Itgives minimum numbers occurrences inrelationship

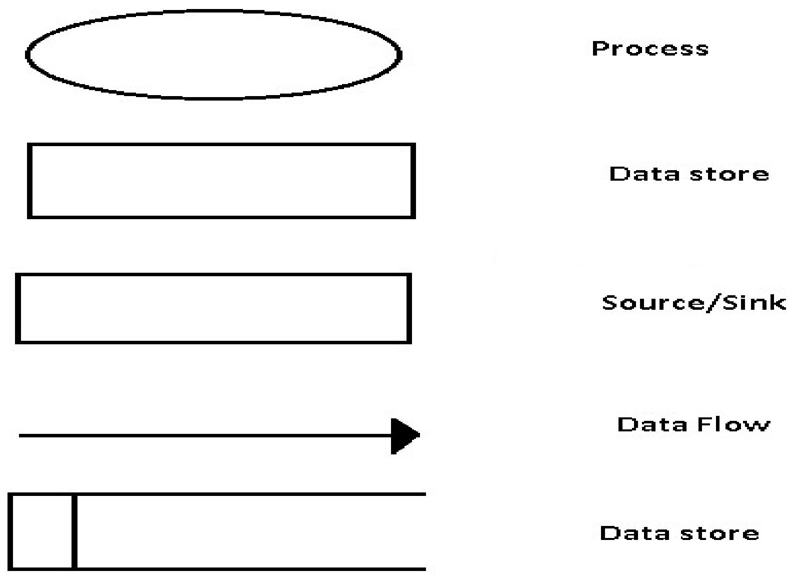
	e) Ans.	**Explain essence of practice.** **Essence of practice:** **Understand the problem (communication and analysis)** - Who has a stake in the solution to the problem? - What are the unknowns (data, function, behavior)? - Can the problem be compartmentalized? - Can the problem be represented graphically? **Plan a solution (planning, modeling and software design)** - Have you seen similar problems like this before? - Has a similar problem been solved and is the solution reusable? - Can sub problems be defined and are solutions available for the sub problems? **Carry out the plan (construction; code generation)** - Does the solution conform to the plan? Is the source code traceable back to the design?	**4M** ***Explana tion essence of practice 1M each***

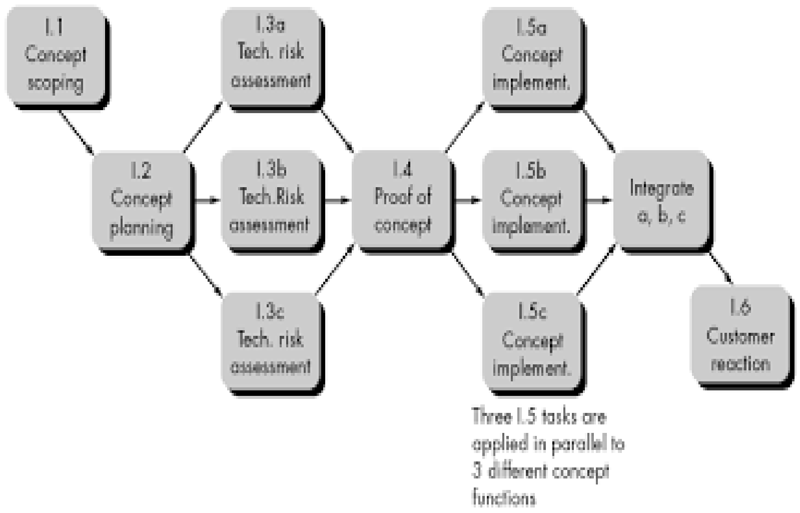
		- Is each component of the solution correct? Has the design and code been reviewed? **Examine the results for accuracy (testing and quality assurance)** - Is it possible to test each component of the solution? - Does the solution produce results that conform to the data, function, and behavior that are required?	
	f) Ans.	**What do you mean by process framework? Explainwith suitable diagram.** 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. 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. Each action in process framework is populated with individual work tasks that accomplish some part of the work implied by the action. Five generic framework activities can be used during the development of small programs, the creations of large web applications and for the engineering of large complex computer-based systems. **1. Communication:** Communication framework activity involves heavy communication and collaboration with the customer, encompasses requirements gathering and other related activities. **2. Planning:** Planning activity establishes a plan for software engineering work that follows. Planning describes the technical tasks to be conducted, the resources that will be required, the risks that are likely the work products to be produced. **3. Modeling:** Modeling activity encompasses the creation of models that allow the developer and the customer to better understand software requirements specifications and the design that will achieve those requirements. **4. Construction:**	**4M** ***Description 2M***

		Construction activity combines code generation and the testing. **5. Deployment:** The software is delivered to the customers who evaluates the delivered product and provides feedback based on the evaluation.  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 **Fig. softwareprocess framework** A software process can be characterized as shown in the diagram 1) A common process framework is established by defining a small number of framework activities that are applicable to all software projects, regardless of their size or complexity. 2) A number of task sets—each a collection of software engineering work tasks, project 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. 3) Umbrella activities—such as software quality assurance, software configuration management, and measurement—overlay the process model. 4) Umbrella activities are independent of any one framework activity	*Diagram 2M*

		and occur throughout the process. 5) Software Engineering Umbrella Activities (Software project tracking and control, Formal technical reviews, Software quality assurance, Software configuration management, Document preparation and production, Reusability management, Measurement, Risk management).	
3.	**a)** Ans.	**Attempt any FOUR of the following:** **Explain software engineering as a layered approach.** Software engineering is a layered technology. The layers of software engineering as shown in the above diagram are:-  **1. A Quality Focus:** Anyengineering 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, documents, data, reports, forms etc.) are produced, milestones are established, quantity is ensured and change is properly	**16** **4M** ***Diagram 2M*** ***Explana tion 2M***

		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.	
	b) Ans.	**Explain following requirement engineering tasks:** **(i) Negotiation** **(ii) Specification.** **(i) Negotiation:** This phase will involve the negotiation between what user actual expects from the system and what is actual feasible for the developer to build. Often it is seen that user always expect lot of things from the system for lesser cost. But based on the other aspect and feasibility of a system the customer and developer can negotiate on the few key aspect of the system and then they can proceed towards the implementation of a system **(ii) Specification**: A specification can be a re-written document, a set of graphical models,a formal mathematical model,and a collection of usage scenario, a prototype, or any combinations of these. The specification is the final work product produced by the requirement engineers. It servesas the foundation for subsequent software engineering activities. It describes the function and performance of a computer based system and the constraints that will govern its development.	**4M** ***2M for negotiation*** ***2M for specification***
	c) Ans.	**What is DFD? Explain its symbol?** 1) Data Flow Diagram (DFD) is also called as Bubble chart‘. This is a graphical technique that represents information flow, and transformer those are applied when data moves from input to output. 2) DFD represents system requirements those becomes program in design. 3) DFD may be further partitioned into different levels to show	**4M** ***2M for explaining DFD***

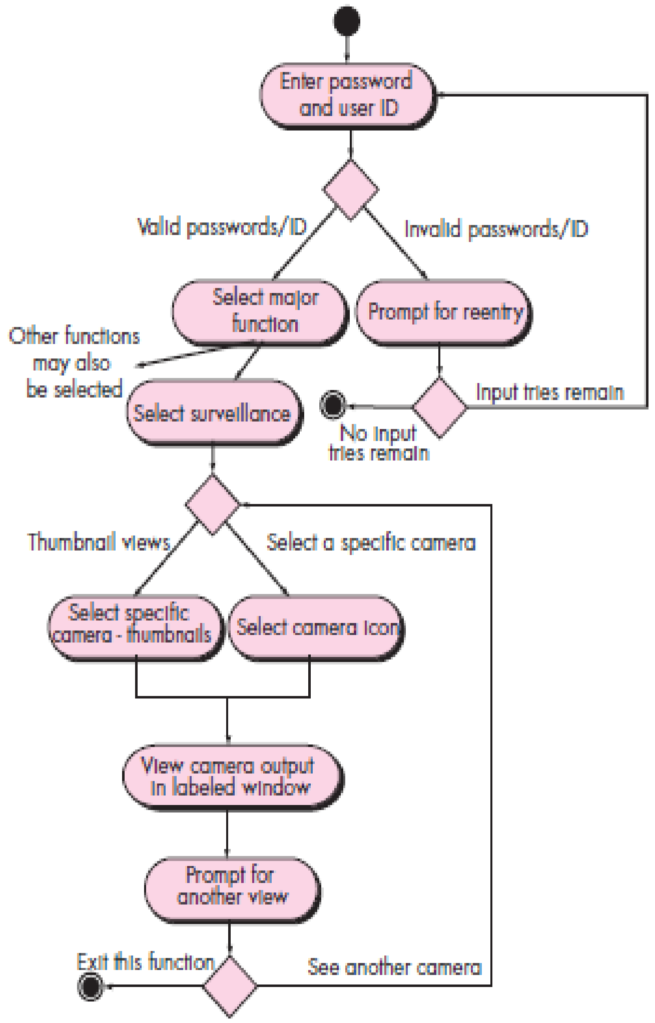
		detailed information flow e.g. level 0, level 1, level 2 etc. 4) DFD focuses on the fact ‗what data flow‘ rather how data is processed 5) DFD is used to represent information flow, and the transformers those are applied when data moves from input to output. 6) To show the data flow with more details the DFD is further extended to level 1, level 2, level 3 etc. as per the requirement. 7) The typical value for the DFD is seven. Any system can be well represented with details up to seventh levels. The symbols that are used are  Process Data store Source/Sink Data Flow Data store	***Any 4 symbols each of ½M diagram and its use in DFD***
	d) Ans,	**How can project scheduling affect Integration testing?** The project schedule provides a road map for a software project manager. If it has been properly developed, the project schedule defines the tasks and milestones that must be tracked and controlled as the project proceeds. When faced with severe deadline pressure, experienced project managers sometimes use a project scheduling and control technique called time-boxing. The time-boxing strategy recognizes that the complete product may not be deliverable by the predefined deadline.	**4M** ***Explaining schedule 1M***

		Therefore, an incremental software paradigm is chosen and a schedule is derived for each incremental delivery. The tasks associated with each increment are then time-boxed. This means that the schedule for each task is adjusted by working backward from the delivery date for the increment. A "box" is put around each task. When a task hits the boundary of its time box (plus or minus 10 percent), work stops and the next task begins. The initial reaction to the time-boxing approach is often negative: "If the work isn'tfinished, how can we proceed?" The answer lies in the way work is accomplished. By the time the time-box boundary is encountered, it is likely that 90 percent of the task has been completed.11 The remaining 10 percent, although important, can (1) be delayed until the next increment or (2) be completed later if required. Rather than becoming "stuck" on a task, the project proceeds toward the delivery date.	***Listing or explain how it affect integration testing 3M***
	e) Ans.	**What is the concept of task network?**  I.1 Concept scoping I.2 Concept planning I.3a Tech. risk assessment I.3b Tech.Risk assessment I.3c Tech. risk assessment I.4 Proof of concept I.5a Concept implement. I.5b Concept implement. I.5c Concept implement. Integrate a, b, c I.6 Customer reaction Three I.5 tasks are applied in parallel to 3 different concept functions Individual tasks and subtasks have interdependencies based on their sequence. In addition, when more than one person is involved in a software engineering project, it is likely that development activities and tasks will be performed in parallel. When this occurs, concurrent tasks must be coordinated so that they will be complete when later tasks require their work product(s).	**4M** ***Diagram 2M*** ***Explain concept of task network 2M***

		A task network, also called an activity network, is a graphic representation of the task flow for a project. It is sometimes used as the mechanism through which task sequence and dependencies are input to an automated project scheduling tool. In its simplest form (used when creating a macroscopic schedule), the task network depicts major software engineering tasks. The concurrent nature of software engineering activities leads to a number of important scheduling requirements. Because parallel tasks occur asynchronously, the planner must determine intertask dependencies to ensure continuous progress toward completion. In addition, the project manager should be aware of those tasks that lie on the critical path. That is, tasks that must be completed on schedule if the project as a whole is to be completed on schedule.	
	f) Ans.	**Explain SCM in short.** Software Configuration Management (SCM) is a set of activities that have been developed to manage change throughout the life cycle of computer software. SCM can be viewed as a software quality assurance activity that is applied throughout the software process. In the sections that follow, we examine major SCM tasks and important concepts that help us to manage change. SCM is an important element of software quality assurance. Its primary responsibility is the control of change. However, SCM is also responsible for the identification of individual SCIs and various versions of the software,the auditing of the software configuration to ensure that it has been properly developed, and the reporting of all changes applied to the configuration. Any discussion of SCM introduces a set of complex questions: • How does an organization identify and manage the many existing versions of a program (and its documentation) in a manner that will enable change to be accommodated efficiently? • How does an organization control changes before and after software is released to a customer? • Who has responsibility for approving and ranking changes? • How can we ensure that changes have been made properly? •What mechanism is used to appraise others of changes that are made?	**4M** ***SCM 1M*** ***If the diagram is drawn 2M along with the explanation 1M*** ***or*** ***SCM 1M Reasons why changes are required 3M***

		These questions lead us to the definition of five SCM tasks: identification, version control,change control, configuration auditing, and reporting.	
4.	**a)** Ans.	**Attempt any <u>FOUR</u> of the following:** **Give possible reasons of why software is delivered late.** Although there are many reasons why software is delivered late, most can be traced to one or more of the following root causes: • An unrealistic deadline established by someone outside the software development group and forced on managers and practitioners within the group. • Changing customer requirements that are not reflected in schedule changes. • An honest underestimate of the amount of effort and/or the number ofresources that will be required to do the job. • Predictable and/or unpredictable risks that were not considered when the project commenced. • Technical difficulties that could not have been foreseen in advance. • Human difficulties that could not have been foreseen in advance. • Miscommunication among project staff that results in delays. • A failure by project management to recognize that the project is falling behind schedule and a lack of action to correct the problem. Aggressive (read "unrealistic") deadlines are a fact of life in the software business. Sometimes such deadlines are demanded for reasons that are legitimate, from the point of view of the person who sets the deadline. But common sense says that legitimacy must also be perceived by the people doing the work.	**16** **4M** ***½ M for each reason total 8 point must be there***
	b) Ans.	**Explain test case design in detail.** The design of tests for software and other engineered products can be as challenging as the initial design of the product itself. Yet,software engineers often treat testing as an afterthought, developing test cases that may "feel right" but have little assurance of being complete. Recalling the objectives of testing, we must design tests that have the highest likelihood of finding the most errors with a minimum amount of time and effort. A rich variety of test case design methods have evolved for software.	**4M** ***Explanation4M***

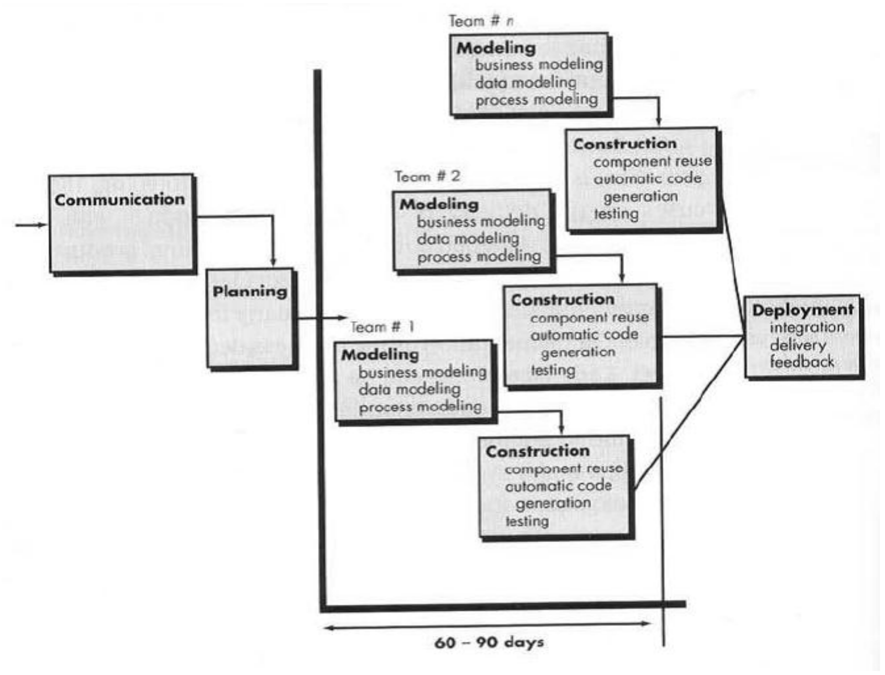
		These methods provide the developer with a systematic approach to testing. More important, methods provide a mechanism that can help to ensure the completeness of tests and provide the highest likelihood for uncovering errors in software. Any engineered product (and most other things) can be tested in one of two ways: (1) Knowing the specified function that a product has been designed to perform, tests can be conducted that demonstrate each function is fully operational while at the same time searching for errors in each function; (2) Knowing the internal workings of a product, tests can be conducted to ensure that "all gears mesh," that is, internal operations are performed according to specifications and all internal components have been adequately exercised. The first test approach is called black-box testing and the second, white-box testing.	
	c) Ans.	**Explain scenario based modeling in detail.** ***(Note: Any other diagram such as use case or swin lane diagram can also be given as example)*** Analysis modeling with UML begins with the creation of scenarios. In scenario Based Modeling the system is represented in user point of view. Scenario based elements are 1. Use case diagram 2. Activity diagram 3. Swim lanes diagram Step 1:First of all, identify the tasks in the project. Step 2:You can add more information to the task boxes, such as who is doing the task and the timeframes. You can add this information inside the box or can add it somewhere near the box. Step 3: Now, arrange the boxes in the sequence that they are performed during the project execution. The early tasks will be at the left hand side and the tasks performed at the later part of the project execution will be at the right hand side. The tasks that can be performed in parallel should be kept parallel to each other (vertically).You may have to adjust the sequence a number of times until you get it right. This is why software is an easy tool for creating activity diagrams.	**4M** ***Listing various scenario based model 1M***

		Step 4:Now, use arrows to join task boxes. These arrows will show the sequence of the tasks. Sometimes, a 'start' and an 'end' box can be added to clearly present the start and the end of the project. Example of activity diagram:Activity diagramfor Access camera surveillancedisplay camera views function The activity diagram is shown for the Function named Access Camera Surveillance-Display Camera Views. The flowof the system is shown in the diagram  Enter password and user ID Valid passwords/ID Invalid passwords/ID Select major function Prompt for reentry Other functions may also be selected Select surveillance Input tries remain No input tries remain Thumbnail views Select a specific camera Select specific camera - thumbnails Select camera icon View camera output in labeled window Prompt for another view Exit this function See another camera	***3M for drawing the diagram and explaining***

	d) Ans.	**What is meant by software deployment?** **Software Deployment:** The deployment phase includes 3 actions namely 1. Delivery 2. Support 3. Feedback 1. The delivery cycle provides the customer and the end user with an operational software increment that provides usable functions and features. 2. The support cycle provides documentation, human assistance for all functions and features introduced during all deployment cycles to date. 3. Each feedback cycle provides the software team with useful inputs. The feedback can help in modifications to the functions, features and even the approach for the next increments. The delivery of the software increment is an important milestone of any software project. A number of key principles should be followed as the team prepares to deliver an increment. **1. Customer expectations for the software must be managed** Before the software delivery the project team should ensure that all the requirements of the users are satisfied. **2. A complete delivery package should be assembled and tested** The system containing all executable software, support data files, tools and support documents should be provided with beta testing at the actual user side. **3. A support regime must be established before the software is delivered** This includes assigning the responsibility to the team members to provide support to the users in case of problem. **4. Appropriate instructional materials must be provided to end users** At the end of construction various documents such as technical manual, operations manual, user training manual, user reference manual should be kept ready. These documents will help in providing proper understanding and assistance to the user. **5. Buggy software should be fixed first, delivered later.** Sometimes under time pressure, the software delivers low-quality increments with a warning to the customer that bugs will be fixed in the next release. Customers will forget you delivered a high-quality	**4M** *Any four principle 1M each*

		product a few days late, but they will never forget the problems that a low quality product caused them. The software reminds them every day.	
	e) Ans.	**What is SRS?** A software requirements specification (SRS) is a complete description of the behavior of the system to be developed. It includes a set of use cases describe all of the interactions that the users will have with the software. In addition to use cases, the SRS contains functional requirements and non functional requirements. Functional requirements define the internal workings of the software: that is, the calculations, technical details, data manipulation and processing, and other specific functionality that shows how the use cases are to be satisfied. Non-functional requirements impose constraints on the design or implementation (such as performance requirements, quality standards, or design constraints). The purpose of SRS document is providing a detailed overview of software product, its parameters and goals. SRS document describes the project's target audience and its user interface, hardware and software requirements. It defines how client, team and audience see the product and its functionality.	**4M** ***SRS 4M***
	f) Ans.	**What is agile process?** Agile programming is an approach to project management, typically used in software development. It helps teams react to the instability of building software through incremental, iterative work cycles, known as sprints. **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.	**4M** ***Explaining agile process 2M*** ***Features of agile 2M***

		Iterative Agile software processes acknowledge that we get things wrong before we get them right. Therefore, theyfocus on short cycles. Within each cycle, a certain set of activities is completed. **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. **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. **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. **People-Oriented** Agile processes favor people over process and technology. They evolve through adaptation in an organic manner. Developers	

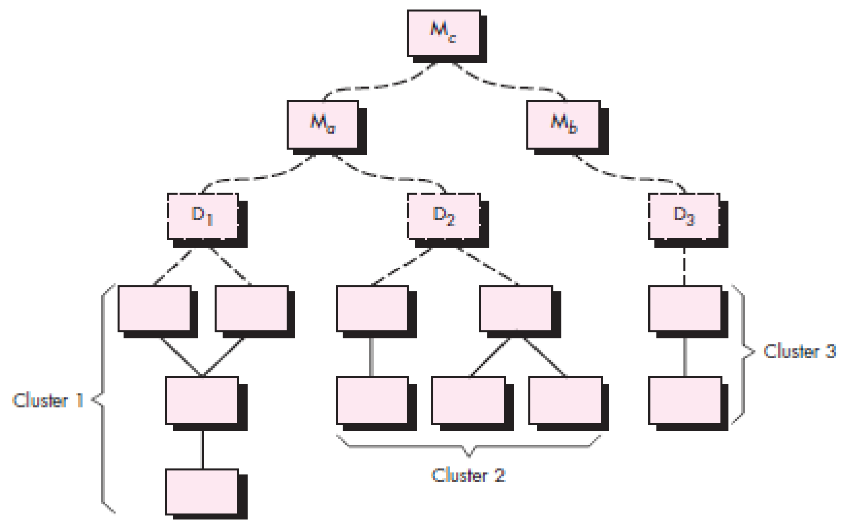
		that are empowered raise their productivity, quality, and performance. **Collaborative** Agile processes foster communication among team members. Communication is a vital part of any software development project. When a project is developed in pieces, understanding how the pieces fit together is vital to creating the finished product.	
5.	**a)** Ans.	**Attempt any <u>TWO</u> of the following:** **Explain RAD model with its advantages and disadvantages.** **The RAD Model:**  Team # n Modeling business modeling data modeling process modeling Construction component reuse automatic code generation testing Team # 2 Communication Planning Team # 1 Deployment integration delivery feedback 60 – 90 days Rapid Application 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	**16** **8M** ***Diagram 2M***

		#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. **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. Due to risks in new approach only modularized systems are recommended through RAD. 5. Suitable for scalable component based systems. **Disadvantages:** 1. Success of RAD model 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.	***Description 2M*** ***Advantages 2M*** ***Disadvantage 2M***
	b) Ans.	**Describe in detail eight principles of good planning.** **Principle 1. Understand the scope of the project.** It's impossible to use a road map if you don't know where you're going. Scope	**8M**

		provides the software team with a destination. **Principle 2. Involve stakeholders in the planning activity.** Stakeholders define priorities and establish project constraints. To accommodate these realities, software engineers must often negotiate order of delivery, time lines, and other project-related issues. **Principle 3. Recognize that planning is iterative.** A project plan is never engraved in stone. As work begins, it is very likely that things will change. As a consequence, the plan must be adjusted to accommodate these changes. In addition, iterative, incremental process models dictate re-planning after the delivery of each software increment based on feedback received from users. **Principle 4. Estimate based on what you know.** The intent of estimation is to provide an indication of effort, cost, and task duration, based on the team's current understanding of the work to be done. If information is vague or unreliable, estimates will be equally unreliable. **Principle 5. Consider risk as you define the plan.** If you have identified risks that have high impact and high probability, contingency planning is necessary. In addition, the project plan (including the schedule) should be adjusted to accommodate the likelihood that one or more of these risks will occur. **Principle 6. Be realistic.** People don't work 100 percent of every day. Noise always enters into any human communication. Omissions and ambiguity are facts of life. Change will occur. Even the best software engineers make mistakes. These and other realities should be considered as a project plan is established. **Principle 7. Adjust granularity as you define the plan.** Granularity refers to the level of detail that is introduced as a project plan is developed. A "high-granularity" plan provides significant work task detail that is planned over relatively short time increments (so that tracking and control occur frequently). A "low-granularity" plan provides broader work tasks that are planned over longer time periods. In general granularity moves from high to low as the project time line moves away from the current date. Over the next few weeks or months, the project can be planned in significant detail. Activities that won't occur for many months do not require high granularity (too much can change). **Principle 8. Define how you intend to ensure quality.** The plan should identify how the software team intends to ensure quality. If	***Any 8 principles with description 1M each***

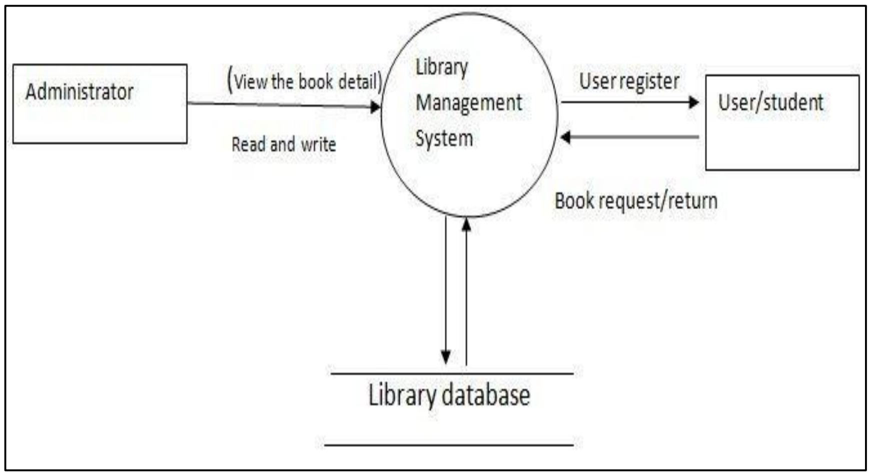
		technical reviews are to be conducted, they should be scheduled. If pair programming is to be used during construction, it should be explicitly defined within the plan. **Principle 9. Describe how you intend to accommodate change.** Even the best planning can be obviated by uncontrolled change. You should identify how changes are to be accommodated as software engineering work proceeds. For example, can the customer request a change at any time? If a change is requested, is the team obliged to implement it immediately? How is the impact and cost of the change assessed? **Principle 10. Track the plan frequently and make adjustments as required.** Software projects fall behind schedule one day at a time. Therefore, it makes sense to track progress on a daily basis, looking for problem areas and situations in which scheduled work does not conform to actual work conducted. When slippage is encountered, the plan is adjusted accordingly.	
	c) Ans.	**What is philosophy of six sigma? Explain Six sigma strategies.** 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 term Six Sigma is derived from six standard deviations instances (defects) per million occurrences—implying an extremely high quality standard. The Six Sigma methodology defines three core steps: • Define customer requirements and deliverables and project goals via well-defined methods of customer communication. • Measure the existing process and its output to determine current quality performance (collect defect metrics). • Analyze 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 the process by eliminating the root causes of defects. • 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	**8M** ***Remark Description 2M*** ***DMADV 3M***

		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 the process to a) Avoid the root causes of defects and b) To meet customer requirements. • Verify that the process model will, in fact, avoid defects and meet customerrequirements. This variation is sometimes called the DMADV (define, measure, analyze, design, and verify) method.	***DMAIC 3M***
6.	**a)** Ans.	**Attempt any <u>FOUR</u> of the following:** **Write meaning of PERT and CPM.** **PERT:** PERT stands for Program Evaluation Review Technique. PERT is a project management technique, used to manage uncertain activities of a project. It is a technique of planning and control of time. It focuses of event. It is a probabilistic model. It is appropriate for high precision time estimate. It manages Unpredictable Activities.PERT chart represent another view of project. It does represent inter task relationships more effectively.Tasks and milestones are included in the chart symbols such as circles; squares are used to depict tasks and milestone. Microsoft uses rectangles to represent task. Each task rectangle is divided into sections with task name at the top and task-id/duration in the middle. **CPM:** CPM stands for critical path method. A project of any kind involves a number of activities. Some of them are interdependent while others are independent. It is important that project management should effectively plan, schedule, co-ordinate and optimize the activities of the various participants in the project. There are certain activities which are to be completed within the stipulated time. If those critical activities are not completed within the prescribed time line, the completion of the whole project is hampered. If the project is quite large effective control over all the activities is difficult. To control such projects, Network techniques have been developed.	**16** **4M** ***PERT 2M*** ***CPM 2M***

	b) Ans.	**Explain the steps of bottom up integration.** **Bottom-Up integration:** This begins with the construction a test of small modules. The components are integrated from the bottom-up, the functionally provided by components subordinate to a given level is always available and the need for stubs is eliminated. A bottom-up integration strategy may be implemented with the following steps: 1. Low-level components are combined into clusters that perform a specific software sub-function. 2. A driver is written to coordinate test case input and output. 3. The cluster is tested 4. Drivers are removed and clusters are combined moving upward in the program structure.  M_c M_a M_b D_1 D_2 D_3 Cluster 1 Cluster 2 Cluster 3	**4M** ***Each step 1M with diagram***
	c) Ans.	**List the objective of black box testing.** 1. Black-box tests are used to demonstrate that software functions are operational 2. Input is properly accepted and output is correctly produced, 3. The integrity of external information (e.g., a database) is maintained. 4. A black-box test examines some fundamental aspect of a system with little regard for the internal logical structure of the software.	**4M** ***1M for each objective***

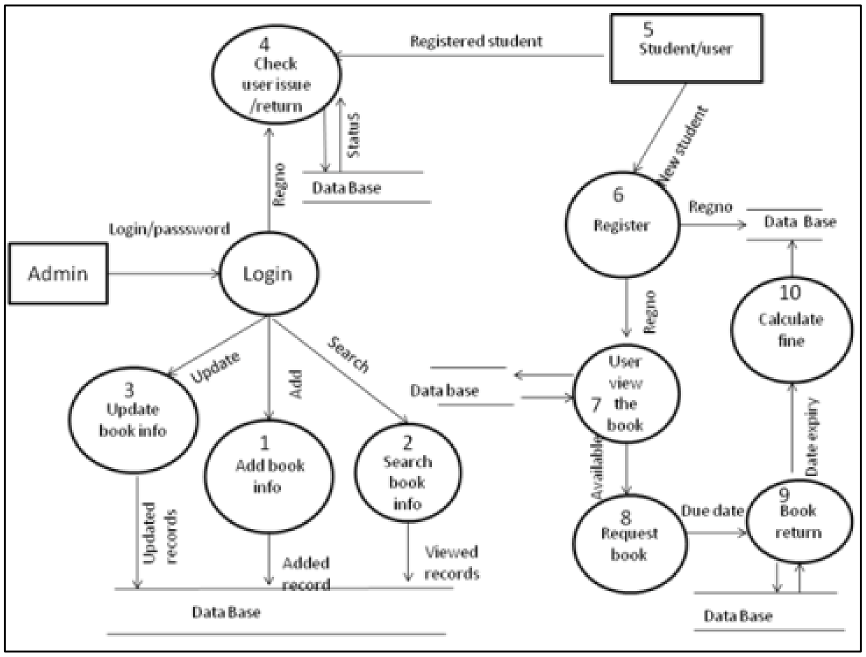
d) **For library management system draw level 0 and level 1 DFD.** **4M**
(Note: ***Any other diagram also shall be considered***

Ans.



Level 0 2M

DFD Level 0 for Library Management System



Level 1 2M

DFD Level 1 for Library Management System

	e) Ans.	**What are the characteristics of good design.** **Design Process:** Software design is an iterative process through which requirements are translated into a **"blueprint"** for constructing the software. The design is representation at a high level of abstraction – data, functional, and behavioral requirements. As design iterations occur, subsequent refinement leads to design representations at much lower levels of abstraction. There are three characteristics that serve as a guide for the evaluation of a good design: 1. The design must implement all the explicit requirements contained in the requirements model, and it must accommodate all the implicit requirements desired by stakeholders. 2. The design must be a readable, understandable guide for those who generate code and for those who test and subsequently support the software. 3. The design should provide a complete picture of the software, addressing the data, functional and behavioral domains for implementation.	**4M** ***1M for Design Process*** ***3Mfor characteristics***
	f) Ans.	**State advantages of PSP and TSP.** **PSP Advantages: -** 1. PSP represents a disciplined, metrics-based approach to software engineering. 2. PSP resulting improvement in software engineering productivity and software quality are significant. 3. It helps the software engineers in developing high quality software products. 4. It guides the engineer for personal improvement. 5. It gives the confidence to do the job the way you know you should. 6. The PSP gives the command over your work. **TSP Advantages: -** 1. Defines roles and responsibilities for each team member. 2. Track quantitative project data. 3. Identifies a team process that is appropriate for the project and a	**4M** ***Any 2 PSP Advantages 2M*** ***Any 2 Advantages TSP 2M***

		strategy for implementing the process; 4. Defines local standards that are applicable to the team's software engineering work; 5. Continually assesses risk and reacts to it; 6. Tracks, manages, and reports project status.	