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 judgment 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.

1 A) Attempt any three of the following: Marks 12

a) What are the core principles of software engineering? Explain.

(For each principle explanation -1 Mark, Any 4(four) principle)

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: KISS (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.

The Third Principle: 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.

The Fourth Principle: 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 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. Specify with an eye to the users. Design, keeping the implementers in mind. Code with concern for those that must maintain and extend the system. Someone may have to debug the code you write, and that makes them a user of your code. Making their job easier adds value to the system.

The Fifth Principle: 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.

The Sixth Principle: 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. To leverage the reuse possibilities that OO programming provides requires forethought and planning. There are many techniques to realize reuse at every level of the system

development process. Those at the detailed design and code level are well known and documented. New literature is addressing the reuse of design in the form of software patterns. However, this is just part of the battle. Communicating opportunities for reuse to others in the organization is paramount. 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.

Seventh Principle: 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.

b) State any four attributes of good software.

(For each attribute - 1 Mark)

Ans:

1. Functionality:

It refers to the degree of performance of the software against its intended purpose.

2. Reliability:

It refers to the ability of the software to provide desired functionality under the given conditions.

3. Usability:

It refers to the extent to which the software can be used with ease and simple.

4. Maintainability:

Software must evolve to meet changing needs.

5. Dependability:

Software must be trustworthy.

6. Efficiency:

Software should not make wasteful of system resources.

7. Acceptability:

Software must accepted by the users for which it was designed.

8. Portability:

It refers to the ease with which software developers can transfer software from one platform to another, without changes.

9. Integrity:

It refers to the degree to which unauthorized access to the software can be prevented.

10. Robustness:

It refers to the degree to which the software can keep on functioning in spite of being provided with invalid data.

c) Explain following terms with the help of example of software engineering:

i) cardinality

ii) relationships

iii) data objects

iv) attributes.

(for each define term- ½ Marks, for their example- ½ Marks)

Ans:

i) Cardinality

Cardinality is the specification of the number of occurrences of one object that can be related to the number of occurrences of another object. Cardinality is usually expressed as simply 'one' or 'many'.

Example: One object can relate to only one other object (a 1:1 relationship);

One object can relate to many other objects (a 1: N relationship);

Some number of occurrences of an object can relate to some other number of occurrences of another object (an M: N relationship);

ii) Relationships

Data object are connected to one another in variety of different ways. This links or connection of data objects or entities with each other is called as relationship.

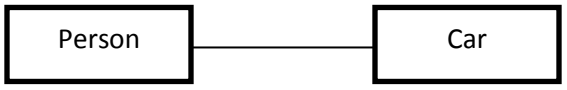
Example:

A connection is established between person and car , because the two objects are related.

1. A person owns a car

2. A person purchase a car
3. A person sells a car
4. person cleans a car

The relationship "owns", "purchase", "sells", and "cleans" define the relevant connections between " person "and "car".



iii) Data objects

A "data object" is a representation of almost any composite information that must be understood by software. By composite information, something that has a number of different properties or attributes.

Example:

"Width" (a single value) would not be a valid data object, but dimensions (incorporating height, width and depth) could be defined as an object.

iv) Attributes

Attributes define the properties of a data object and take one of three different characteristics.

They can be used to:

1. Name an instance of the data objects,
2. Describe the instance,
3. Make reference to another instance in another table.

Example: attributes must be defined as "identifier". Referring to data object "car", a reasonable identifier or attribute might be the "ID No"," Color".

d) What do you mean by process framework? Explain with suitable diagram.

(Description – 2 Marks, Diagram – 2 Marks)

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 fig 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:

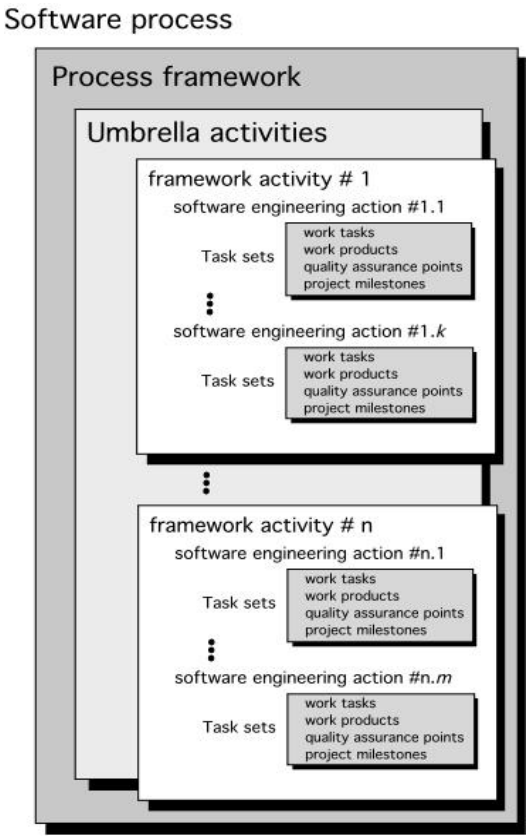
Modeling activity encompasses the creation of models that allow the developer and the customer to better understand software requirements specifications and the design that will achieve those requirements.

4. Construction:

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 framework

B) Attempt any one of the following: Marks 06

a) Explain DFD with example.

(Description - 3 Marks, Example - 3 Marks, Any relevant example should be consider)

Ans: DFD is refined into greater levels of detail; the analyst performs an implicit functional decomposition of the system, thereby accomplishing the fourth operational analysis principle for function.

At the same time, the DFD refinement results in a corresponding refinement of data

(1) The level 0 data flow diagram should depict the software/system as a single bubble;

(2) Primary input and output should be carefully noted;

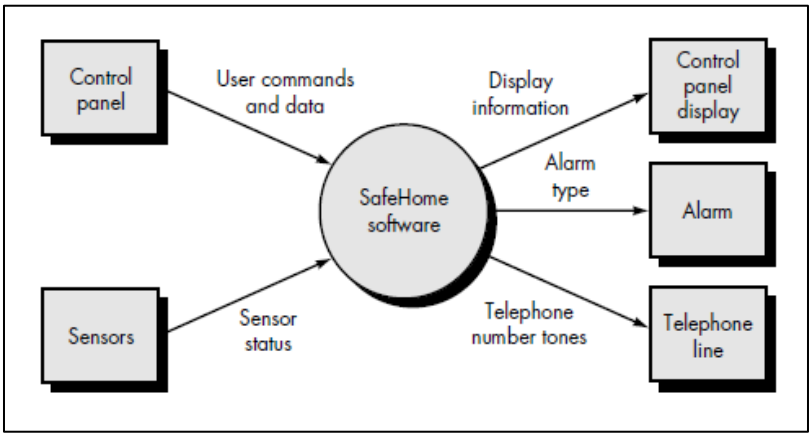
(3) Refinement should begin by isolating candidate processes, data objects, and stores to be represented at the next level;

(4) All arrows and bubbles should be labeled with meaningful names;

(5) Information flow continuity must be maintained from level to level, and

(6) One bubble at a time should be refined.

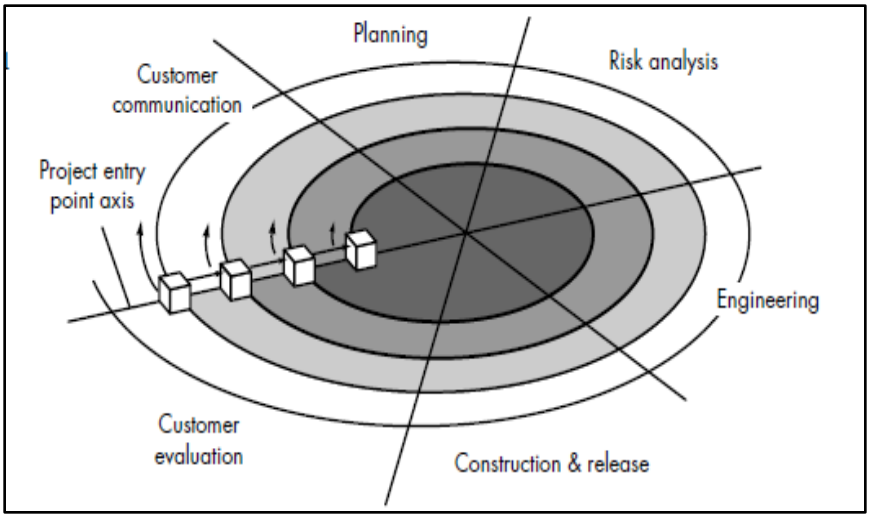
Considering the Safe Home product, a level 0 DFD for the system is shown in Figure. The primary external entities (boxes) produce information for use by the system and consume information generated by the system. The labeled arrows represent data objects or data object type hierarchies. For example, user commands and data encompass all configuration commands, all activation/deactivation commands, all miscellaneous interactions, and all data that are entered to qualify or expand a command.



b) Explain different tasks of regions of spiral model with diagram.

(Diagram - 2 Marks, Explanation - 4 Marks)

Ans:



Traditional method or model of software development

Also encompasses all the essential development phases:

- Requirements analysis
- Design
- Code
- Test
- Maintenance

Explicitly addresses the issue of quality assurance by performing the development process in a "step-wise refinement" method.

Each step produces a "deliverable" which embodies the structure or sequential nature of the process. This process naturally focuses on a high degree of customer or stakeholder involvement during the development process

In this diagram, "**Communication**" refers to the Requirements and analysis process, **"Planning**" corresponds to preliminary design and scheduling, **"Modeling"** is the detailed design, **"Construction"** refers to code/debug/integration, and **"Deployment"** is the delivery to the customer and the feedback process.

Software is developed in a series of evolutionary releases

o During early iterations, the release might be a paper model or prototype

o During later iterations, increasingly more complete versions of the engineered system are produced

o The final iteration produces the complete software product

First circuit around the spiral might result in the development of the product specification; might result in a CoDR review by the customer

Next iteration might produce a prototype, containing the GUI, for example; the customer might want to see this, so there could be a PDR and/or CDR at this time

Third time around might be used to fill in more detailed functionality, and release a preliminary working model

Fourth circuit might result in a complete alpha release, which the customer could ―hammer on‖ for a while to test robustness and provide feedback to the solution provider about the product's strengths and weaknesses

Fifth iteration might be a beta test, or it could be the final build for initial release (if the previous circuit was satisfactory enough to warrant this)

Problem with the spiral model: may not appear controllable to the customer, particularly if the customer is more accustomed to the waterfall model.

2 Attempt any four of the following: Marks 16

a) What are the modeling practices in software engineering? Explain their principles.

(Modeling practices types - 1 Mark, for each principle - 1 Mark (consider any 3 principles))

Ans: At a technical level, software engineering begins with a series of modeling tasks that lead to complete specification of requirements and a comprehensive design representation for the software to be built.

The analysis model is a set of models which is nothing but the technical representation of system.

When entity is software, your model must take a different form. At technical levels, software engineering begins with series of modeling tasks that lead to a complete specification of requirements and a comprehensive design representation for the software to be built.

The two dominating analysis modeling techniques are:

1.”structured analysis”

2.”Object-Oriented analysis”

In software engineering two types of models are created as:

i) Analysis models:

They represent the customers/users requirements depicting the software in 3 domains:

A) Information domain

b) Function domain

c) Behavioral domain

ii) Design models:

They represent characteristics of the software that help practitioners to construct it effectively.

a) The architecture

b) The user interface and

c) Component-levels details.

Following are various Analysis Modeling Principles

1. The information domain of a problem must be represented and understood.

– define data objects

– describe data attributes

– establish data relationships

2. The functions that the software is to perform must be defined.

– Identify functions that transform data objects

3. The behavior of the software (as a consequence of external events) must be represented.

– indicate different states of the system

– specify events that cause the system to change state

4. The models that depict information function and behavior must be partitioned in a manner that uncovers detail in a layered (or hierarchical) fashion.

– Refine each model to represent lower levels of abstraction

– refine data objects

– create a functional hierarchy represent behavior at different levels of detail

5. The analysis process should move from essential information toward implementation

Following are various Design Modeling Principles

Principle #1:- **Design should be traceable to an analysis model**.

The analysis mode describes the information domain of the problem, user visible functions and various things about system. The design model translates this information into architecture: a set of sub systems that implement major functions, and a set of component level designs that are the realization of analysis classes.

Principle #2: - Always consider the architecture of the system to be built

Software architecture is a skeleton of the system to be built. It affects interfaces, data structures, program control flow and behavior, the manner in which testing can be conducted, the maintainability of the resultant system and much more.

Principle #3: - Design of data is as important as design of processing functions

Data design is an essential element of architectural design. The manner in which data objects are realized within the design cannot be left to chance. A well-structured data design helps to simplify

program flow, makes the design and implementation of software components easier, an d makes overall processing more efficient.

Principle #4: - Interfaces must be designed with care

The manner in which data flows between the components of a system has much to do with processing efficiency, error propagation, and design simplicity. A well designed interface makes integration easier and assists the tester in validating component functions.

Principle #5: - User interface design should be tuned to the needs of the end-user

In every case it should stress ease of user. The user interface is the visible manifestation of the software. No matter how sophisticated its internal functions, no matter how comprehensive its data structures, no matter how well designed its architecture, a poor interface design often le ads to the perception that the software is bad.

Principle #6: - Component level design should be functionally independent. Functional independence is a measure of the ―single mindedness‖ of a software component. The functionality that is delivered by a component should be cohesive – that is it should focus on one and only one function or sub -function.

Principle #7: - Component should be loosely coupled to one another and to the external environment. Coupling is achieved in many ways – via a component interface, by messaging, through global data. As the level of coupling increases, the likelihood or error propagation also increases and the overall maintainability of the software decreases.

Principle #8: - Design representation should be easily understandable the purpose of design is to communicate information to practitioners who will generate code, to those who will test the software, and to others who may maintain the software in the future. If the design is difficult to understand, it will not serve as an effective communication medium.

Principle #9 The design should be developed iteratively. With each iteration the designer should strive for greater simplicity. Like almost all creative activities, design occurs iteratively. The first iterations works to refine the design and correct errors, but later iterations should strive to make the design as simple as possible.

b) Describe 4 layers of software engineering.

(Description - 3 Marks, Diagram - 1 Mark)

Ans: **Software Engineering – A Layered Technology approach.**

Software engineering is a layered technology. Referring to Figure any engineering approach (including software engineering) must rest on an organizational commitment to quality. Total quality management and similar philosophies foster a continuous process improvement culture, and this culture ultimately leads to the development of increasingly more mature approaches to software engineering. The bedrock that supports software engineering is a quality focus. 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 for a set of *key process areas* that must be established for effective delivery of software engineering technology. The key process areas form the basis for management control of software projects and establish the context in which technical methods are applied, work products (models, documents, data, reports, forms, etc.) are produced, milestones are established, quality is ensured, and change is properly managed.

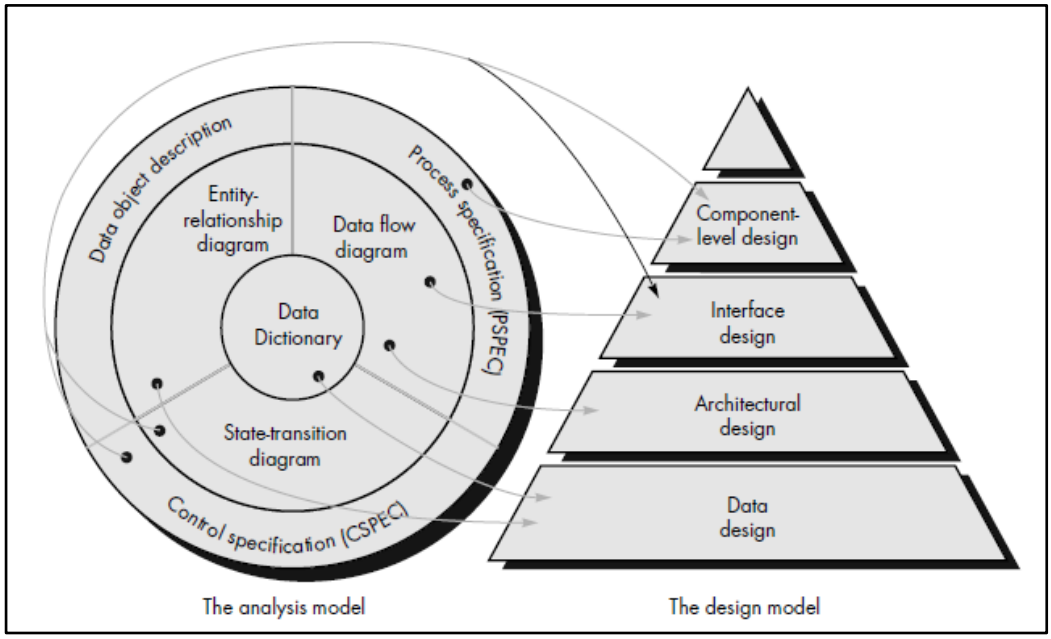
Software engineering methods provide the technical how-to's for building software. Methods encompass a broad array of tasks that include requirements analysis, design, program construction, testing, and support. Software engineering methods rely on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques. 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.

c) With neat diagram explain the translation of analysis model into design model.

(Description - 2 Marks, Diagram - 2 Marks)

Ans:



Software design is applied regardless of the software process model that is used. Beginning once software requirements have been analyzed and specified, software design is the first of three technical activities –design, code generation, and test- that are required to build and verify the software. Each activities transforms information in manner that ultimately results in validated computer software.

Each of the elements of the analysis model provides information that is necessary to create the four models required for a complete specification of design.

1. The **data/class design** transforms analysis classes into design classes along with data structure required for a complete specification of design.
2. The **architectural design** defines the relationship between major structural elements of the software; architectural styles and design patterns help achieve the requirements defined for the system

3. The **interface design** describes how the software communicates with systems that interoperate with it and with humans that use it.
4. The **component-level design** transforms structural elements of the software architecture into procedural description of software components.

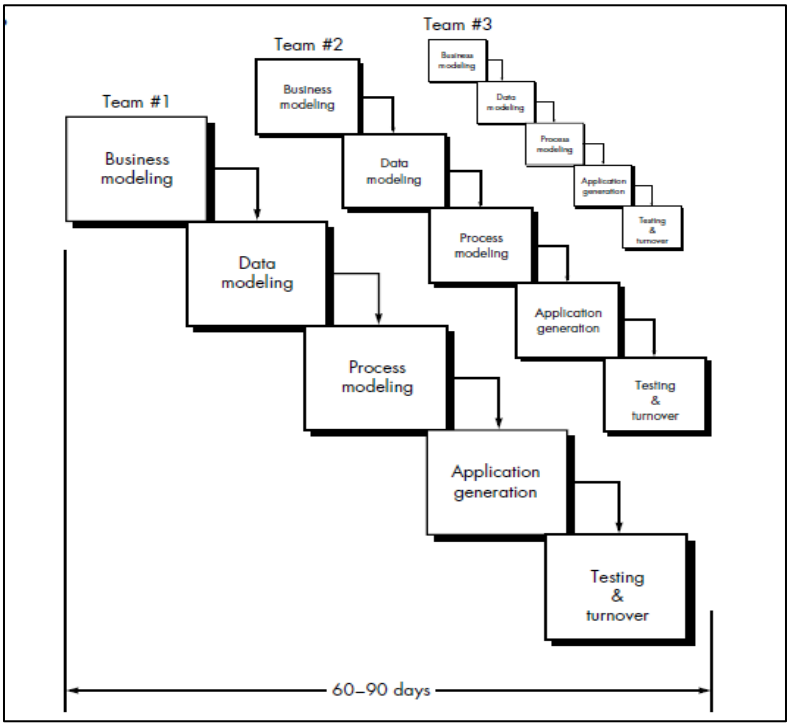
   The flow of information during software design is illustrated in fig.

d) Describe the RAD process model with neat diagram and its advantages.

(Description -2 Marks, Diagram -1 Mark, Advantages - 1 Mark)

Ans:

RAD Model



Rapid application development (RAD) is an incremental software development process model that emphasizes an extremely short development cycle. The RAD model is a ―high-speed‖ adaptation of the linear sequential model in which rapid development is achieved by using component-based construction. If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a ―fully functional system‖ within very short time periods (e.g., 60 to 90 days). Used primarily for information systems applications, the RAD approach encompasses the following phases.

Business modeling: - The information flow among business functions is modeled in a way that answers the following questions: What information drives the business process? What information is generated? Who generates it? Where does the information go? Who processes it?

Data modeling: - The information flow defined as part of the business modeling phase is refined into a set of data objects that are needed to support the business. The characteristics (called *attributes*) of each object are identified and the relationships between these objects defined. Data modelling is considered in.

Process modeling: - The data objects defined in the data modeling phase are transformed to achieve the information flow necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.

Application generation: - RAD assumes the use of fourth generation techniques (Section 2.10). Rather than creating software using conventional third generation programming languages the RAD process works to reuse existing program components (when possible) or create reusable components (when necessary). In all cases, automated tools are used to facilitate construction of the software.

Testing and turnover: - Since the RAD process emphasizes reuse, many of the program components have already been tested. This reduces overall testing time. However, new components must be tested and all interfaces must be fully exercised.

Advantages:-

1. Faster implementation of Project
2. Parallel implementation
3. Projects divided into small teams results into better implementation

e) Write importance of analysis modeling.

(For each importance – 1 Mark, any four (4 relevant importance shall be consider))

Ans:

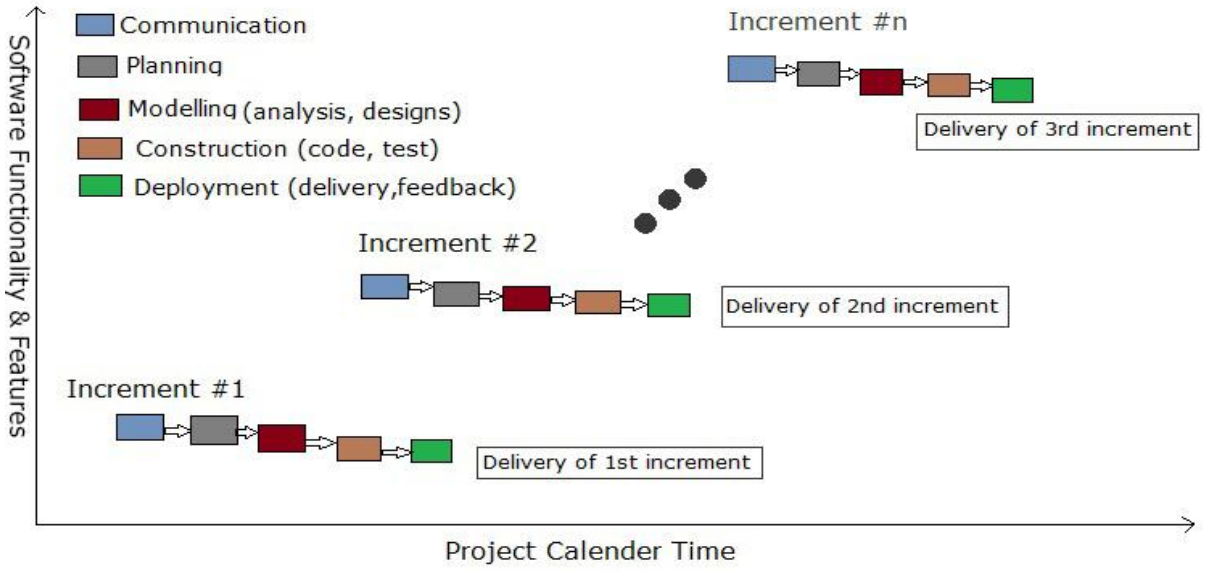
- Designing gets easier to the designer
- Better understanding of system can be accomplished
- System feasibility can be determined
- Defines data objects
- Describes data attributes

- Establishes data relationships
- Identifies functions that transform data objects
- Indicates different states of the system
- Specifies events that cause the system to change state
- Refines each model to represent lower levels of abstraction
- Refines data objects
- Creates a functional hierarchy represent behavior at different levels of detail

f) Explain incremental process model using suitable diagram.

(Description - 2 Marks, Diagram - 2 Marks)

Ans:



1. Incremental model is also called as iterative enhancement model.
2. In this model the software is built in an incremental fashion.
3. Fig shows the project is divided into small subsets called as increments and are implemented by individually.
4. The incremental model combines elements of the linear sequential model with the iterative philosophy of prototyping,
5. Each linear sequence produces deliverable increments of the software.

6. The incremental model is iterative in nature. When an incremental model is used, the first increments are often a "core product."
7. Thai is basic requirements are addressed, but many supplementary features remain undelivered. Incremental development is useful when staffing is unavailable for a complete implementation by the business deadline that has been established for the project.
8. In this model the product is designed, implemented, integrated and tested as a series of incremental builds.
9. Each build consist of code pieces from various modules interacting to provide a specific functional capability.

Example:

Word processing software developed using the incremental paradigm might deliver basic file management editing and document production in the first increment, more sophisticated editing and document production capabilities in second increment;

Spelling and grammar checking in third increment and advanced page layout capability in fourth increment and it should be noted that the process flow for any increment can incorporate the prototyping paradigm.

3 Attempt any four of the following: Marks 16

a) Write difference between cardinality and modaling.

[Note: modaling is to be considered as a modality]

(Any Four relevant Points are expected- 1 Mark each Comparison Point)

Ans:

Cardinality	Modality
Cardinality defines the range of object-to object Relationships	Modality indicates whether or not a relationship between objects is mandatory
Expected Values are 1:1 , 1:N, N:M	Expected values are 0 or 1 only
It does not, however, provide an indication of whether or not a particular data object must participate in the relationship.	It provides indication of participation in the relationship by having values as 1, if value is 0 no participation in relationship will exist.
It gives maximum numbers occurrences in relationship	It gives minimum numbers occurrences in relationship

b) What are different data design element and architectural design elements?

(Data Design Element - 2 Marks, Architectural Design Elements - 2 Marks)

Ans: Data Design Elements

Like other software engineering activities, data design (sometimes referred to as data architecting) creates a model of data and/or information that is represented at a high level of abstraction (the customer/user's view of data). This data model is then refined into progressively more implementation-specific representations that can be processed by the computer-based system. In many software applications, the architecture of the data will have a profound influence on the architecture of the software that must process it.

The structure of data has always been an important part of software design. At the program component level, the design of data structures and the associated algorithms required to manipulate them is essential to the creation of high-quality applications. At the application level, the translation of a data model (derived as part of requirements engineering) into a database is pivotal to achieving the business objectives of a system. At the business level, the collection of information stored in disparate databases and reorganized into a "data warehouse" enables data mining or knowledge discovery that can have an impact on the success of the business itself. In every case, data design plays an important role.

Architectural Design Elements

The architectural design for software is the equivalent to the floor plan of a house. The floor plan depicts the overall layout of the rooms; their size, shape, and relationship to one another; and the doors and windows that allow movement into and out of the rooms. The floor plan gives us an overall view of the house. Architectural design elements give us an overall view of the software.

The architectural model is derived from three sources: (1) information about the application domain for the software to be built; (2) specific requirements model elements such as data flow diagrams or analysis classes, their relationships and collaborations for the problem at hand; and (3) the availability of architectural styles and patterns.

The architectural design element is usually depicted as a set of interconnected subsystems, often derived from analysis packages within the requirements model. Each subsystem may have its own architecture (e.g., a graphical user interface might be structured according to a preexisting architectural style for user interfaces).

c) What is requirement engg. ? What is its need? What are different subtasks included in it?
(Requirement Engineering-1 Mark, Need - 1 Mark, Subtask - 2 Marks)

Ans: Requirement Engineering:-

Software process perspective, requirements engineering is a major software engineering action that begins during the communication activity and continues into the modeling activity. It must be adapted to the needs of the process, the project, the product, and the people doing the work.

Need of Requirement Engineering:-

Requirements engineering tools assist in requirements gathering, requirements modeling, requirements management, and requirements validation.

Subtasks included Requirement Engineering:-

- Inception
- Elicitation
- Elaboration
- Negotiation
- Specification
- Validation
- Requirements management

d) Describe communication principles statements.

(Any 4 Communication Principles are expected - 1 Mark for each statement with description, Only List of Principles 2 Marks)

Ans: **Principle 1 Listen**: Try to focus on the speaker's words, rather than formulating your response to those words. Ask for clarification if something is unclear, but avoid constant interruptions. Never become contentious in your words or actions (e.g., rolling your eyes or shaking your head) as a person is talking.

Principle 2 Prepare before you communicate: Spend the time to understand the problem before you meet with others. If necessary, perform some research to understand business domain jargon. If you have responsibility for conducting a meeting, prepare an agenda in advance of the meeting.

Principle 3 Someone should facilitate the activity: Every communication meeting should have a leader (a facilitator) to keep the conversation moving in a productive direction, (2) to mediate any conflict that does occur, and (3) to ensure than other principles are followed.

Principle 4 Face-to-face communication is best: But it usually works better when some other representation of the relevant information is present. For example, a participant may create a drawing or a "strawman" document that serves as a focus for discussion.

Principle 5 Take notes and document decisions: Things have a way of falling into the cracks. Someone participating in the communication should serve as a "recorder" and write down all important points and decisions.

Principle 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 collaborations serves to build trust among team members and creates a common goal for the team.

Principle 7 Stay focused; modularize your discussion: The more people involved in any communication, the more likely that discussion will bounce from one topic to the next. The facilitator should keep the conversation modular; leaving one topic only after it has been resolved.

Principle 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.

Principle 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 participate should recognize that many topics require discussion and that "moving on" is sometimes the best way to achieve communication agility.

Principle 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.

e) What are PSP and TSP frame work activities? Explain their meaning in detail.

(PSP Framework activities 2 Marks, TSP Framework activities 2 Marks)

Ans: PSP Framework Activities includes:-

The PSP model defines five framework activities:

Planning: This activity isolates requirements and develops both size and resource estimates. In addition, a defect estimates (the number of defects projected for the work) is made. All metrics are recorded on worksheets or templates. Finally, development tasks are identified and a project schedule is created.

High-level design: External specifications for each component to be constructed are developed and a component design is created. Prototypes are built when uncertainty exists. All issues are recorded and tracked.

High-level design review: Formal verification methods (Chapter 21) are applied to uncover errors in the design. Metrics are maintained for all important tasks and work results.

Development: The component-level design is refined and reviewed. Code is generated, reviewed, compiled, and tested. Metrics are maintained for all important tasks and work results.

Postmortem: Using the measures and metrics collected (this is a substantial amount of data that should be analyzed statistically), the effectiveness of the process is determined. Measures and metrics should provide guidance for modifying the process to improve its effectiveness.

TSP Framework Activities includes:-

TSP defines the following framework activities: **project launch, high-level design, implementation, integration and test,** and **postmortem.**

Project Launch: It reviews core objective and describes the TSP structure and content. It assigns terms and roles to developers and describes the customer needs statement. It also establishes team and individual goals.

High Level Design: It creates high-level design, specifies the design, and inspects the design develop an integration test plan.

Implementation: Implementation uses the TSP to implement modules/unit, creates a detailed design of modules/units, reviews the design, translates the design to code, review the code, compile and test the modules/units and analyze the quality of the modules/units.

Integration and Test: Testing builds and integrates these builds into a system. It conducts a system test and produce user documentation.

Postmorterm: It conducts a postmortem analysis, writes a cycle report and produce peer and team evaluations.

4 A) Attempt any three of the following: Marks 12

a) State benefits of ISO standards.

(Any 4 Benefits - 1 Mark each benefit, Any relevant benefits shall be consider)

Ans:

- Well defined and documented procedures improve the consistency of output
- Quality is constantly measured
- Procedures ensure corrective action is taken whenever defects occur
- Defect rates decrease
- Defects are caught earlier and are corrected at a lower cost
- Defining procedures identifies current practices that are obsolete or inefficient
- Documented procedures are easier for new employees to follow
- Organizations retain or increase market share, increasing sales or revenues
- Improved product reliability
- Better process control and flow
- Better documentation of processes
- Greater employee quality awareness
- Reductions in product scrap, rewords and rejections.

b) Differentiate validation and verification.

(Any 4 Points of comparison shall be consider - 1 Mark each comparison point)

Ans:

Validation	Verification
Validation is a dynamic mechanism of validating and testing the actual product.	Verification is a static practice of verifying documents, design, code and program.
It always involves executing the code.	It does not involve executing the code.
It is computer based execution of program.	It is human based checking of documents and files.
Validation uses methods like black box (functional) testing, gray box testing, and white box (structural) testing etc.	Verification uses methods like inspections, reviews, walkthroughs, and Desk-checking etc.
Validation is to check whether software meets the customer expectations and requirements.	Verification is to check whether the software conforms to specifications.
It can catch errors that verification cannot catch. It is High Level Exercise.	It can catch errors that validation cannot catch. It is low level exercise.
Target is actual product-a unit, a module, a bent of integrated modules, and effective final product.	Target is requirements specification, application and software architecture, high level, complete design, and database design etc.
Validation is carried out with the involvement of testing team.	Verification is done by QA team to ensure that the software is as per the specifications in the SRS document.
It generally follows after verification.	It generally comes first-done before validation.

c) Explain briefly :

i) Unit testing

ii) System testing

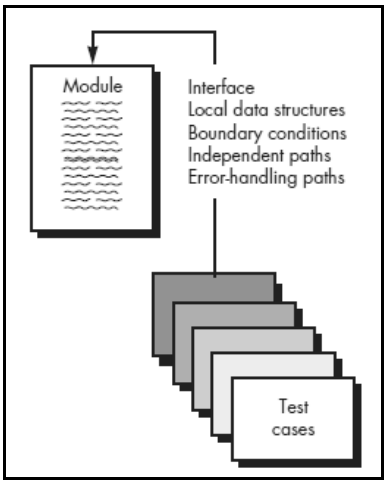
(Unit Testing - 2 Marks, System Testing- 2 Marks, Description of types of System testing is optional; any figure of unit testing shall be consider)

Ans:

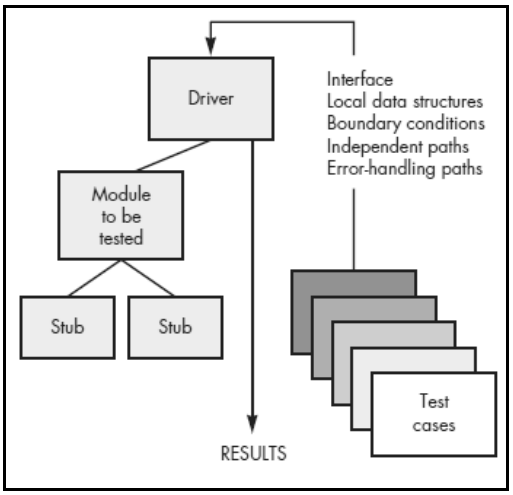
i. Unit Testing

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

structures within the boundaries of a component. This type of testing can be conducted in parallel for multiple components.



O
R



Unit Testing Unit Testing

ii.System Testing

System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all work to verify that system elements have been properly integrated and perform allocated functions.

Following are the types of System Testing

Recovery Testing: Recovery testing is a system test that forces the software to fail in a variety of ways and verifies that recovery is properly performed. If recovery is automatic (performed by the system itself), re-initialization, check-pointing mechanisms, data recovery, and restart are evaluated for correctness. If recovery requires human intervention, the mean-time-to-repair (MTTR) is evaluated to determine whether it is within acceptable limits.

Security Testing: Security testing attempts to verify that protection mechanisms built into a system will, in fact, protect it from improper penetration. "The system's security must, of course, be tested for invulnerability from frontal attack—but must also be tested for invulnerability from flank or rear attack."

Stress Testing: Stress testing executes a system in a manner that demands resources in abnormal quantity, frequency, or volume. For example, (1) special tests may be designed that generate ten

interrupts per second, when one or two is the average rate, (2) input data rates may be increased by an order of magnitude to determine how input functions will respond, (3) test cases that require maximum memory or other resources are executed, (4) test cases that may cause thrashing in a virtual operating system are designed, (5) test cases that may cause excessive hunting for disk-resident data are created. Essentially, the tester attempts to break the program.

Performance Testing: Performance tests are often coupled with stress testing and usually require both hardware and software instrumentation. That is, it is often necessary to measure resource utilization (e.g., processor cycles) in an exacting fashion. External instrumentation can monitor execution intervals, log events (e.g., interrupts) as they occur, and sample machine states on a regular basis. By instrumenting a system, the tester can uncover situations that lead to degradation and possible system failure.

d) What is SCM? What is its need? What are its features?

(SCM - 1 Mark, Need of SCM- 1 Mark, Any two Features of SCM - 2 Marks)

Ans: SCM

When you build computer software, change happens. And because it happens, you need to manage it effectively. Software configuration management (SCM), also called change management, is a set of activities designed to manage change by identifying the work products that are likely to change, establishing relationships among them, defining mechanisms for managing different versions of these work products, controlling the changes imposed, and auditing and reporting on the changes made.

Need of SCM

1. To Identify all items that define the software configuration
2. To Manage changes to one or more configuration items
3. To Facilitate construction of different versions of a software application
4. To Ensure that software quality is maintained as configuration evolves.

Features of SCM:-

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.

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.

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.

B) Attempt any one of the following: Marks 06

a) Explain risk refinement.

(Meaning - 1 Mark, Need of Risk refinement - 1 Mark, CTC - 1 Mark, Sub-conditions- 1 Mark each)

Ans: During early stages of project planning, a risk may be stated quite generally. As time passes and more is learned about the project and the risk, it may be possible to refine the risk into a set of more detailed risks, each somewhat easier to mitigate, monitor, and manage. One way to do this is to represent the risk in condition-transition-consequence

(CTC) format. That is, the risk is stated in the following form:

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

Using the CTC format for the reuse risk one could write:

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

This general condition can be refined in the following manner:

Sub-condition 1: Certain reusable components were developed by a third party with no knowledge of internal design standards.

Sub-condition 2: The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.

Sub-condition 3: Certain reusable components have been implemented in a language that is not supported on the target environment.

The consequences associated with these refined sub-conditions remain the same

(i.e., 30 percent of software components must be custom engineered), but the refinement helps to isolate the underlying risks and might lead to easier analysis and response.

b) Describe six sigma a strategy.

(Basic introduction of Six Sigma - 2 Marks, DMADV - 2 Marks, DMAIC - 2 Marks)

Ans: 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 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 (1) avoid the root causes of defects and (2) to meet customer requirements.

Verify that the process model will, in fact, avoid defects and meet customer requirements.

This variation is sometimes called the DMADV (define, measure, analyze, design, and verify) method.

5 Attempt any two of the following: Marks 16

a) Describe process of CMMI techniques.

(For CMMI definition - 2 Marks, For explanation of each Level - 1 Mark)

Ans: CMMI (Capability Maturity Model Integration) is a proven industry framework to improve product quality and development efficiency for both hardware and software

Sponsored by US Department of Defense in cooperation with Carnegie Mellon University and the Software Engineering Institute (SEI)

Representation as:

- Staged
- Continuous

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.

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.

b) What are the principles used for project scheduling? Explain their meaning.

(Each principle explanation - 1 Mark)

Ans: 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 are 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.

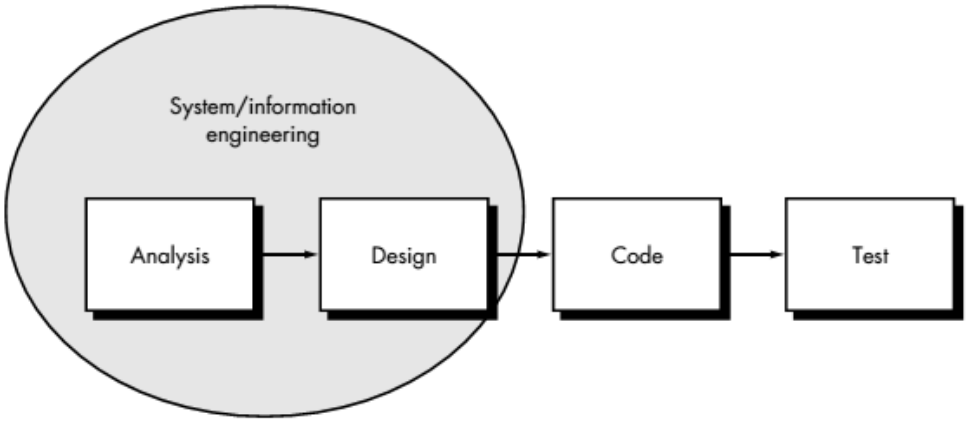
Defined milestones

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) Explain different cycles in software development.

(Figure for - 2 Marks and cycle description for - 6 Marks, Any software process model shall be considered)

Ans: Sometimes called the classic life cycle or the waterfall model, the linear sequential model suggests a systematic, sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing, and support. Figure illustrates the linear sequential model for software engineering. Modeled after a conventional engineering cycle, the linear sequential model encompasses the following activities



System/information engineering and modeling. Because software is always part of a larger system (or business), work begins by establishing requirements for all system elements and then allocating some subset of these requirements to software. This system view is essential when software must interact with other elements such as hardware, people, and databases. System engineering and analysis encompass requirements gathering at the system level with a small

amount of top level design and analysis. Information engineering encompasses requirements gathering at the strategic business level and at the business area level.

Software requirements analysis. The requirements gathering process is intensified and focused specifically on software. To understand the nature of the program(s) to be built, the software engineer ("analyst") must understand the information domain for the software, as well as required function, behavior, performance, and interface. Requirements for both the system and the software are documented and reviewed with the customer.

Design: Software design is actually a multistep process that focuses on four distinct attributes of a program: data structure, software architecture, interface representations, and procedural (algorithmic) detail. The design process translates requirements into a representation of the software that can be assessed for quality before coding begins. Like requirements, the design is documented and becomes part of the software configuration.

Code generation: The design must be translated into a machine-readable form.

The code generation step performs this task. If design is performed in a detailed manner, code generation can be accomplished mechanistically.

Testing: Once code has been generated, program testing begins. The testing process focuses on the logical internals of the software, ensuring that all statements have been tested, and on the functional externals; that is, conducting tests to uncover errors and ensure that defined input will produce actual results that agree with required results.

Support: Software will undoubtedly undergo change after it is delivered to the customer (a possible exception is embedded software). Change will occur because errors have been encountered, because the software must be adapted to accommodate changes in its external environment (e.g., a change required because of a new operating system or peripheral device), or because the customer requires functional or performance enhancements. Software support/maintenance reapplies each of the preceding phases to an existing program rather than a new one.

6 Attempt any four of the following: Marks 16

a) Differentiate between alpha and beta testing.

(Any four (4) points -1 Mark each)

Ans:

ALPHA TESTING	BETA TESTING
Alpha Testing Conducted at Developer Site by End user.	Beta Testing is conducted at User site by End user.
Alpha Testing is Conducted in Control Environment as Developer is present	Beta Testing is conducted in Un-control Environment as Developer is Absent
Less Chances of Finding an error as Developer usually guides user.	More Chances of Finding an error as Developer can use system in any way
It is kind of mock up testing	The system is tested as Real application
Error/Problem may be solved in quick time if possible.	The user has to send difficulties to the developer who then corrects it.
Short process	Lengthy Process

b) Describe integration testing approaches

i) Top-down integration

ii) Bottom up integration.

(For explanation top-down integration - 2 Marks, For explanation bottom-up integration- 2 Marks)

Ans:

i) Top- down integration

1. Main control module used as a test driver and stubs are substitutes for components directly subordinate to it.

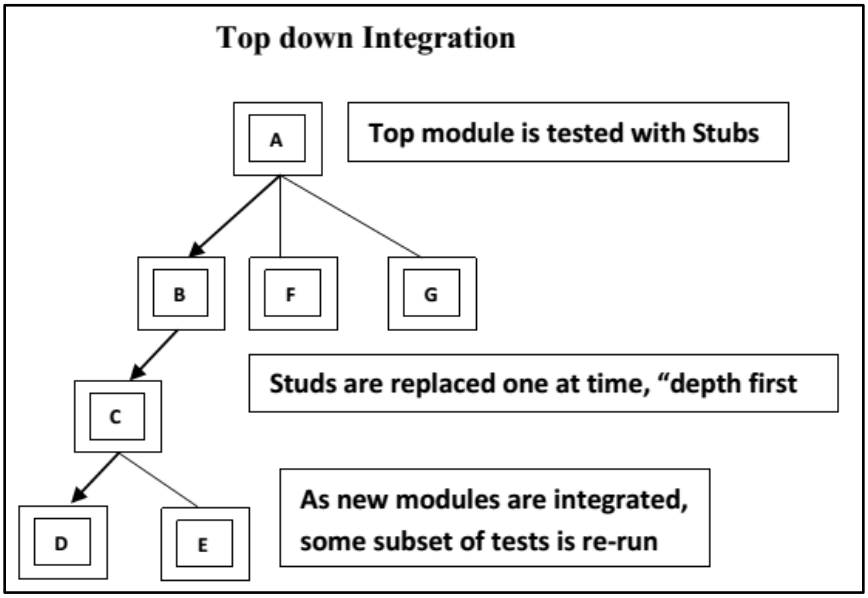
2. Subordinate stubs are replaced one at a time with real components (following the depth-first or breadth-first approach).

3. Tests are conducted as each component is integrated.

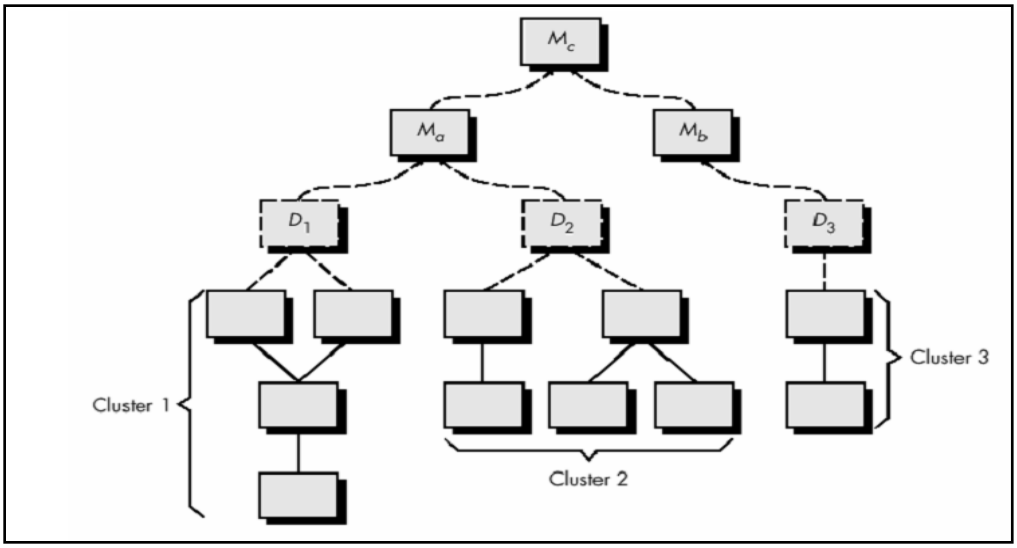
4. On completion of each set of tests and other stub is replaced with a real component.

5. Regression testing may be used to ensure that new errors not introduced.

ii) Bottom- up integration



1. Low level components are combined in clusters that perform a specific software function.
2. A driver (control program) 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.



c) Explain characteristics of software testing strategy.

(Each Characteristics with explanation - 1 Mark, Only list - 2 Marks)

Ans:

1. **A good test has a high probability of finding an error**. To achieve this goal, the tester must understand the software and attempt to develop a mental picture of how the software might fail. Ideally, the classes of failure are probed. For example, one class of potential failure in a GUI (graphical user interface) is a failure to recognize proper mouse position. A set of tests would be designed to exercise the mouse in an attempt to demonstrate an error in mouse position recognition.
2. **A good test is not redundant.** Testing time and resources are limited. There is no point in conducting a test that has the same purpose as another test. Every test should have a different purpose (even if it is subtly different). For example, a module of the SafeHome software (discussed in earlier chapters) is designed to recognize a user password to activate and deactivate the system. In an effort to uncover an error in password input, the tester designs a series of tests that input a sequence of passwords. Valid and invalid passwords (four numeral sequences) are input as separate tests. However, each valid/invalid password should probe a different mode of failure.
3. **A good test should be "best of breed".** In a group of tests that have a similar intent, time and resource limitations may mitigate toward the execution of only a subset of these tests. In such cases, the test that has the highest likelihood of uncovering a whole class of errors should be used.
4. **A good test should be neither too simple nor too complex**. Although it is sometimes possible to combine a series of tests into one test case, the possible side effects associated with this approach may mask errors. In general, each test should be executed separately.

d) Describe people factor in software management spectrum.

(Description - 3 Marks, Various teams associated with project -1 Mark)

Ans: The People

The people factor is so important that the Software Engineering Institute has developed a people management capability maturity model (PM-CMM), to enhance the readiness of software

organizations to undertake increasingly complex applications by helping to attract, grow, motivate, deploy, and retain the talent needed to improve their software development capability.

The people management maturity model defines the following key practice areas for software people: recruiting, selection, performance management, training, compensation, career development, organization and work design, and team/culture development. Organizations that achieve high levels of maturity in the people management area have a higher likelihood of implementing effective software engineering practices. The PM-CMM is a companion to the software capability maturity model that guides organizations in the creation of a mature software process.

Following are the various categories of people associated with the project.

- The Stakeholders: - This include Senior managers, Project (technical) managers, Practitioners, Customers, End user
- Team Leaders: - Leaders of various teams associated with the project
- The Software Team:- Entire software team
- Agile Teams: - Temporary teams associated with software
- Coordination and Communication Issues

e) Enlist the reason for failure of software project.

(Each Point-1 Mark (Any four))

Ans: Although there are many reasons why software is delivered late, most can be traced to one or more of the following root causes:

1. An unrealistic deadline established by someone outside the software development group and forced on managers and practitioners within the group.
2. Changing customer requirements that are not reflected in schedule changes.
3. An honest underestimate of the amount of effort and/or the number of resources that will be required to do the job.
4. Predictable and/or unpredictable risks that were not considered when the project commenced.
5. Technical difficulties that could not have been foreseen in advance.
6. Human difficulties that could not have been foreseen in advance.
7. Miscommunication among project staff that results in delays.
8. A failure by project management to recognize that the project is falling behind schedule and a lack of action to correct the problem.