MAHARASHTRA STATE BOARD OF TECHNICAL EDUCATION
(Autonomous)
(ISO/IEC - 27001 - 2005 Certified)

WINTER-15 EXAMINATION

Model Answer

Subject Code: 17513 **Subject Name: Software Engineering**

Important Instructions to examiners:

1) The answers should be examined by key words and not as word-to-word as given in Themodel 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 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.

Marks
12

1. (A) Answer any THREE of the following:

(a) Describe the characteristics of software
(Each Characteristics - 1 Marks; Diagram is optional)

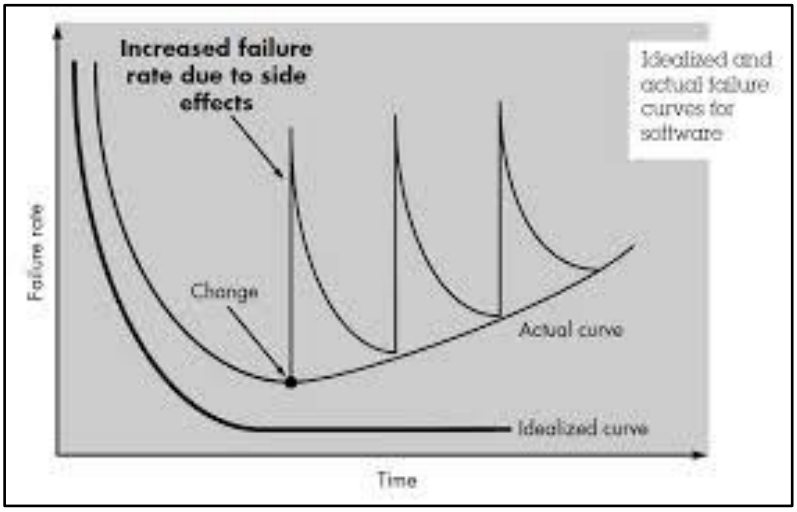
Ans:

Software is written to handle an Input – Process – Output system to achieve predetermined goals. Software is logical rather than a physical system element. Therefore software has following characteristics:

1. Software is developed or engineered; it is not manufactured in the classical sense.
2. Software doesn't "wear out" like hardware and it is not degradable over a period.
3. Although the industry is moving toward component – based construction, most software continues to be custom built.
4. A software component should be designed and implemented so that it can be reused in many different programs.

5. Software is not susceptible to the environmental maladies that cause hardware to wear out. Hence, the failure rate curve for software should take the form of "idealized curve" as shown in the above figure. Undiscovered defects will cause high failure rates early in the life of a program. However, these are corrected and the curve flattens as shown. Hence the software doesn't wear out, but it does deteriorate.

Failure curve for Software:



(b) Briefly explain software engineering as a layered technology.
(Each layer - 1 Mark)

Ans:

Software engineering is a layered technology. The layers of software engineering as shown in the above diagram are:-

1. **A Quality Focus:**
   Any engineering approach (including software engineering) must rest on an organizational commitment to quality. Total quality management, six sigma and similar philosophies foster a continuous process improvement culture, and it is this culture that ultimately leads to the development of increasingly more effective approaches to software engineering. The bedrock that supports software engineering is a quality focus.
2. **Process Layer:**
   The foundation for software engineering is the process layer. Software Engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software. Process defines 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 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.

(c) With reference to requirement engineering, explain
(i) Inception and
(ii) Elicitation
(Inception - 2 Marks; Elicitation - 2 Marks)

Ans:

(i) Inception

Inception means beginning. It is always problematic to the developer that from where to start.

The customer and developer meet and they decide overall scope and nature of the problem.

The aim is

a. To have the basic understanding of problem
b. To know the people who will use the software
c. To know exact nature of problem.

(ii) Elicitation

Elicitation means to draw out the truth or reply from anybody. In relation with requirement engineering, elicitation is a task that helps the customer to define what is required. To know the objectives of the system to be developed is a critical job.

a. Problem of Scope:

The boundary of the system is ill-defined or the customers/users specify unnecessary technical detail that may confuse, rather than clarify, overall system objectives.

b. Problem of understanding

The customers/users are not completely sure of what is needed, have a poor understanding of the capabilities and limitations of their computing environment, don't have full understanding of the problem domain, have trouble communicating needs to the system engineer, omit information that is believed to be "obvious", specify requirements that conflict with the needs of other customer/users or specify requirements that ambiguous or untestable.

c. Problems of volatility

Volatility means change from one state to another. The customer's requirement may change time to time.

(d) With reference to software design give the meanings of

(i) Modularity

(ii) Functional independence

(iii) Refactoring

(iv) Information hiding

(1 Mark for each)

Ans:

1. Modularity :

Software architecture and design patterns embody modularity; that is, software is divided into separately named and addressable components, sometimes called modules that are integrated to satisfy problem requirements.

It is the compartmentalization of data and function. It is easier to solve a complex problem when you break it into manageable pieces. "Divide-and-conquer". Don't over-modularize. The simplicity of each small module will be overshadowed by the complexity of integration "Cost".

2. Functional Independence :

The concept of functional Independence is a direct outgrowth of modularity and the concepts of abstraction and information hiding.

Design software such that each module addresses a specific sub-function of requirements and has a simple interface when viewed from other parts of the program structure. Functional independence is a key to good design, and to software quality. Independence is assessed using two qualitative criteria: cohesion and coupling.

3. Refactoring :

It is a reorganization technique that simplifies the design of a component without changing its function or behavior. When software is re-factored, the existing design is examined for redundancy, unused design elements, inefficient or unnecessary algorithms, poorly constructed data structures, or any other design failures that can be corrected to yield a better design.

4. Information Hiding :

Hiding implies that effective modularity can be achieved by defining a set of independent modules that communicate with one another only that information necessary to achieve software function.

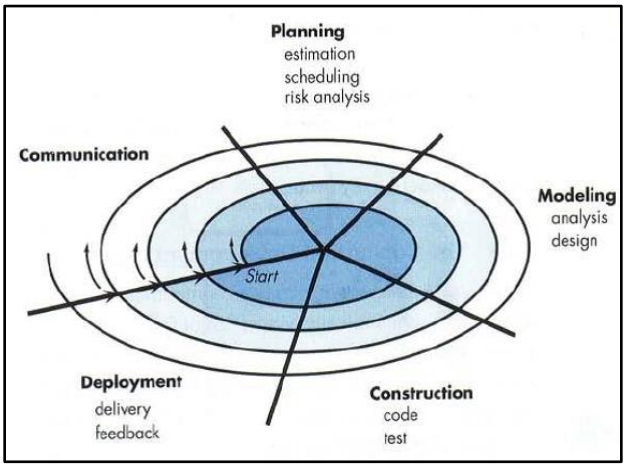
The use of Information Hiding as a design criterion for modular systems provides the greatest benefits when modifications are required during testing and later, during software maintenance. Because most data and procedures are hidden from other parts of the software, inadvertent errors introduced during modifications are less likely to propagate to other location within the software.

(B) Answer any ONE of the following: **6**

(i) With a neat diagram, explain the nature and general steps of spiral model. Also give its advantages and disadvantages.
(Diagram - 2 Marks; Description - 2 Marks; Advantages - 1 Mark; disadvantage -1 Mark)

Ans:

The Spiral Model:



Boehm (1988) viewed the software development process in light of risks involved. Spiral model combines development activities with risk management to minimize and control the risk impact.

It is an evolutionary model which couples iterative nature of prototyping with controlled and systematic aspects of the waterfall model. It also provides scope for RAD for increasingly complete software.

The spiral development model is a risk-driven process model generator that is used to guide multi-stakeholder concurrent engineering of software intensive systems. From the figure given above, a spiral model is divided into a set of framework activities defined by the software engineering team. As this evolutionary process begins, the software team performs activities that are implied by a circuit around the spiral in a clockwise direction, beginning at the center. Risk is considered as each revolution is made. Anchor point milestones – a combination of work products and conditions that are attained along the path of the spiral – are noted for each evolutionary pass.

Each pass through the planning region results in adjustments to the project plan. Cost & schedule are adjusted based on feedback derived from the customer after delivery. In addition, the project manager adjusts the planned number of iterations required to complete the software.

The initial circuit around the spiral can be for the concept development and with multiple iterations. The spiral traverses outward for new product development spiral development remains operative for the life span of software. This may be a realistic approach for large scale software development. As the process progresses both users and developers better understand the system. The spiral model can be adopted to apply throughout the life cycle of an application, from concept development to maintenance.

Advantages:

1. One is a cyclic approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk.

2. The set of anchor point milestones for ensuring stakeholder commitment to obtain feasible and mutually satisfactory system solutions.

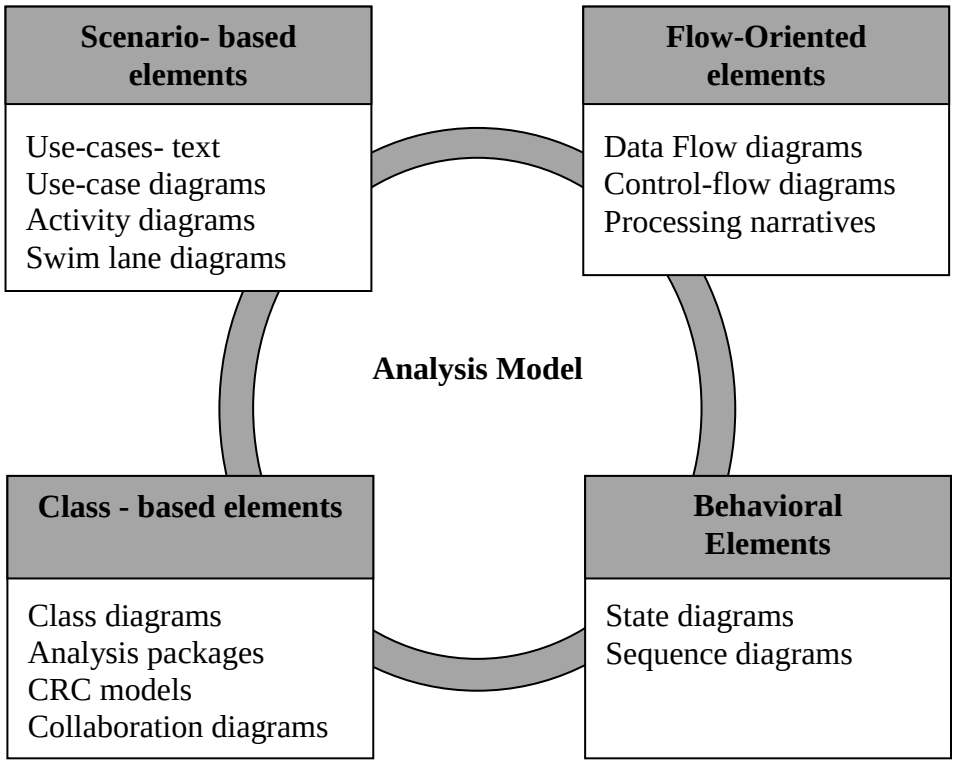
Limitations:

1. The system demands risks identification and monitoring to prevent hurdles.

2. System can get into infinite iterations.

(ii) Explain the various elements of analysis modeling in detail.
(List of various elements – 2 Marks; Description of each element – 1 Mark)

Ans:



Elements of the analysis model **(Diagram Optional)**

Scenario based Elements

The system is described from the user's point of view using this approach. This is often the first part of analysis model that is developed to serve as input for the creation of other modeling elements.

Class-based Elements

Each usage scenario implies a set of objects that are manipulated as an actor interacts with the system. These objects are categorized into classes – a collection of things that have similar attributes and common behaviors.

Behavioral Elements

The behavior of the system can have profound effect on the design that is chosen. The analysis model must provide modeling elements that depict the behavior. The state diagram is one of the methods for representing behavior of a system.

Flow-Oriented Elements

The information is transformed as it flows through the computer based system. The system accepts inputs in a variety of forms, applies functions to transform it; and produces output in different forms. The transforms may comprise a single logical comparison, a complex numerical algorithm or an expert system. The elements of the information flow are included here.

2. Answer any FOUR of the following: 16

(a) Define PSP and TSP. Give advantages of TSP.
(Definition of PSP - 1 Mark; Definition of TSP - 1 Mark; Advantages - 2 Marks)

Ans:

Personal Software Process (PSP):

Watts Humphrey suggests that in order to change an ineffective personal process, an individual must move through four phases, each requiring training and careful instrumentation. The personal software process (PSP) emphasizes personal measurement of both the work product that is produced and the resultant quantity of the work product. In addition, the PSP makes the practitioner responsible for project planning (e.g. estimating and scheduling) and empowers the practitioner to control the quantity of all software work products that are developed.

The PSP process model defines five framework activities: Planning, high-level design review, development and Postmortem.

Team Software Process(TSP):

The goal of TSP is to build a "Self-directed" project team that organizes itself to produce high-quantity software. Humphrey defines the following objectives for TSP:

- Build self-directed teams that plan and track their work, establish goals, and own their processes and plans. These can be pure software teams or integrated product teams (IPT) of 3 to about 20 engineers.
- Show managers how to coach & motivate their teams and how to help them sustain peak performance.
- Accelerate software process improvement by making CMM level 5 behavior normal and expected.
- Provide improvement guidance to high-maturity organizations.
- Facilitate university teaching of industrial-grade team skills.

Advantages of TSP:

A self-directed team has a consistent understanding of its overall goals and objectives. To form a self-directed team, we must collaborate well internally and communicate well externally.

Like PSP, TSP is a rigorous approach to software engineering that provides distinct and quantifiable benefits in productivity and quality.

(b) Explain the features of Agile software development approach.
(Any four feature – 1 Mark each)

Ans:

Agile programming is an approach to project management, typically used in software development. It helps teams react to the instability of building software through 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.

Iterative

Agile software processes acknowledge that we get things wrong before we get them right. Therefore, they focus on short cycles. Within each cycle, a certain set of activities is completed.

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.

(c) In which situation RAD model is applicable? Give its advantages and disadvantages.
(Description - 2 Marks; Any 2 Advantages - 1 Mark; Any 2 Disadvantages - 1 Mark)

Ans:

The RAD Model:

The RAD model approach is applicable, 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.

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 #n leading to each module getting developed almost simultaneously.

Advantages:

1. Changing requirements can be accommodated and progress can be measured.
2. Powerful RAD tools can reduce development time.
3. Productivity with small team in short development time and quick reviews risk control increases reusability of components, better quality.
4. Risk of new approach only modularized systems are recommended through RAD.
5. Suitable for scalable component based systems.

Limitations:

1. RAD model success depends on strong technical team expertise and skills.
2. Highly skilled developers needed with modeling skills.
3. User involvement throughout life cycle. If developers & customers are not committed to the rapid fire activities necessary to complete the System in a much-abbreviated time frame, RAD projects will fail.

May not be appropriate for very large scale systems where the technical risks are high.

(d) Describe the principles of deployment.
(Each Principle - 1 Mark; any four principle)

Ans:

The key principles of deployment are:

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 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) Explain PSPEC with an example.
(Description - 2 Marks; Example -2 Marks; any relevant description/ Diagram shall be considered)

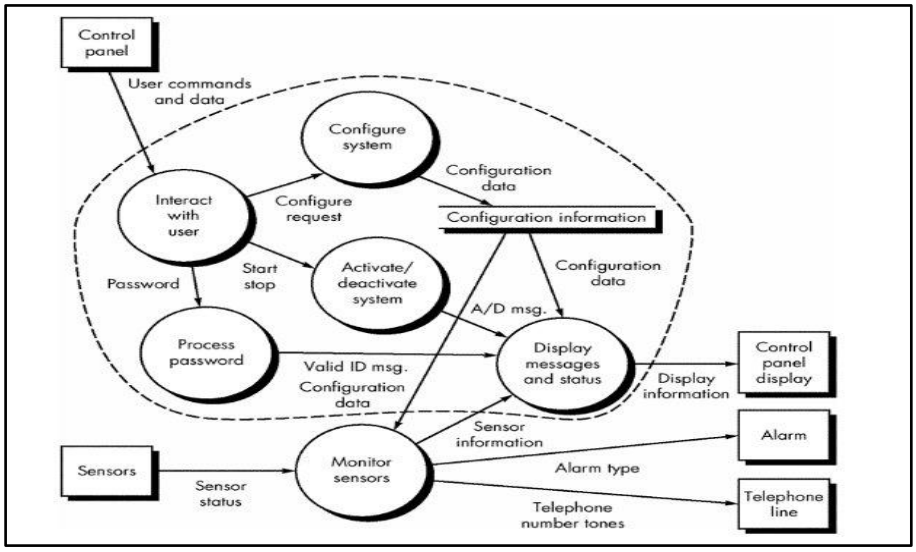
Ans:

Creating Process Specification (PSPEC)

This process is used to describe all flow model processes that appear at the final level of refinement. The content of the process specification can include narrative text, a program design language (PDL) description of the process algorithm, mathematical equations,

tables, or Unified Modeling Language (UML) activity diagrams. By providing a PSPEC to accompany each bubble in the flow model, the software engineer creates a mini-SPEC that can serve as a guide to the design of the software component that will implement the process.

Example: To illustrate the use of the PSPEC, consider the process password transform represented in the flow model for Safe Home. The PSPEC for this function might take the form.



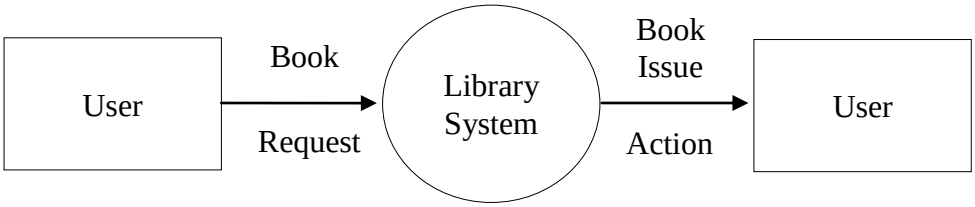
The process password transform performs password validation at the control panel for the SafeHome security function. Process password receives a four-digit password from the interact with user function. The password is first compared to the master password stored within the system. If the master password matches, [valid id message = true] is passed to the message and status display function. If the master password does not match, the four digits are compared to a table of secondary passwords (these may be assigned to house guests and/or workers who require entry to the home when the owner is not present). If the password matches an entry within the table,[valid id message=true] is passed to the

message and status display function. If there is no match,[valid id message=false] is passed to the message and status display function.

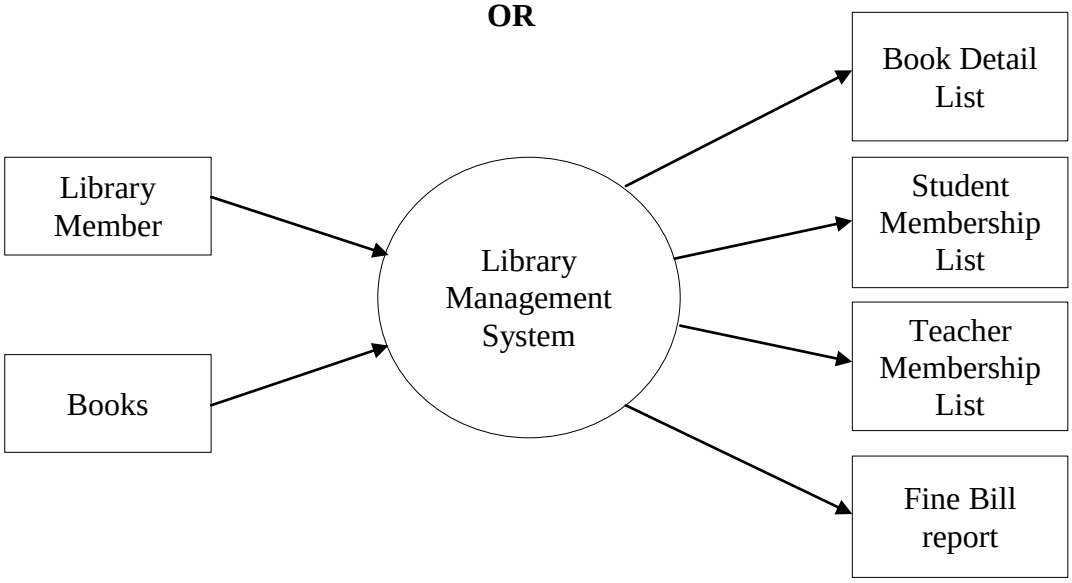
(f) Draw level '0'and '1' DFD for library management system. Make suitable assumptions.
(Level 0 - 2 Marks; Level 1 - 2 Marks; any other relevant diagram shall also be considered)

Ans:

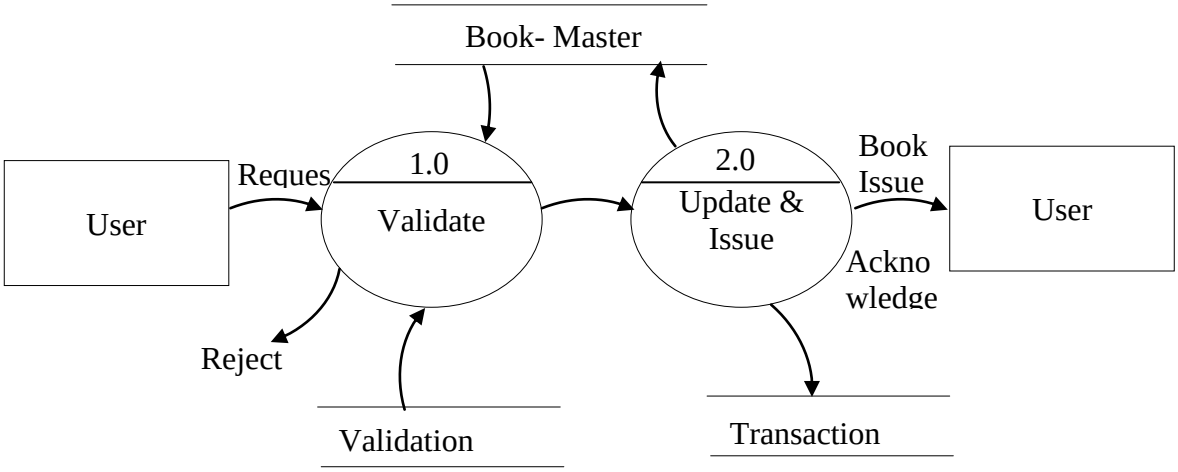
Level 0 DFD for Library Management System



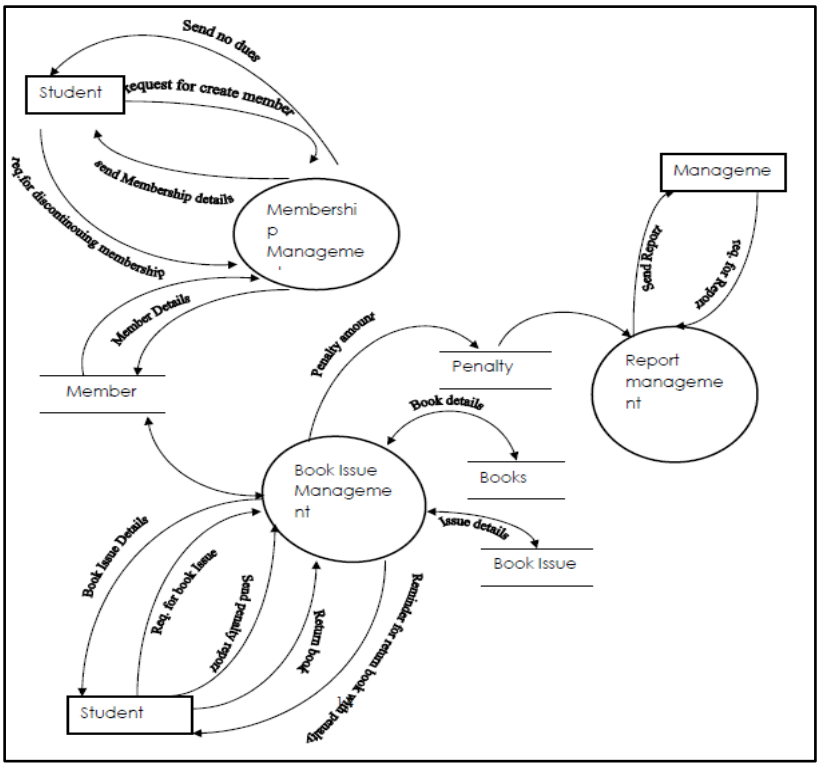
OR



Level 1 DFD for Library Management System



OR



3. Answer any FOUR of the following: **16**

(a) Explain the basic process framework activities.
(4 Activities - 1 Mark Each)

Ans:

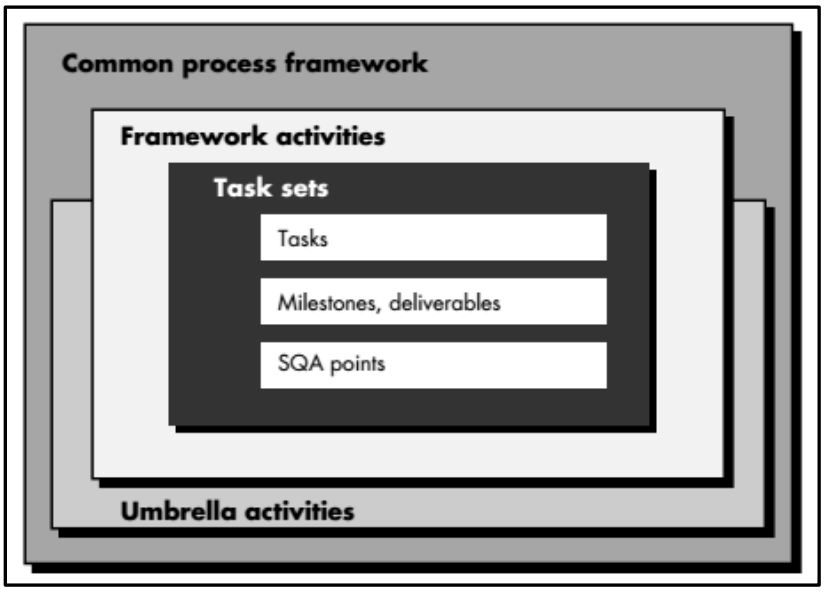
Basic Process Framework Activities:

1. Software Process: A software process can be characterized as shown in Figure 1.5. 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. Software Product: 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. Basic Framework Activities: Finally, umbrella activities—such as software quality assurance, software configuration management, and measurement to overlay the process model.

4. Umbrella Activities: Umbrella activities are independent of any one framework activity and occur throughout the process.



OR

A generic process framework for software engineering encompasses five activities:

Communication: Before any technical work can commence, it is critically important to communicate and collaborate with the customer (and other stakeholders). The intent is to understand stakeholders' objectives for the project and to gather requirements that help define software features and functions.

Planning: Any complicated journey can be simplified if a map exists. A software project is a complicated journey, and the planning activity creates a "map" that helps guide the

team as it makes the journey. The map—called a software project plan—defines the software engineering work by describing the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.

Modeling: In this phase the software team prepares a model of intended system. Team makes design of algorithm, flowcharts and other designs of system are made in this phase. The developer then ensures they develop system as per the design provided to them. The modeling team refers outcome of earlier model such as communication, planning and designs it as per the requirement.

Construction: This activity combines code generation (either manual or automated) and the testing that is required uncovering errors in the code.

Deployment: The software (as a complete entity or as a partially completed increment) is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation.

(b) Briefly describe the principles of communication.
(Any 4 Principles -1 Mark each)

Ans:

Communication practice is two way communication between the client and the developer, hence it is also known as requirement elicitation.

1. **Listen carefully**
   i. To collect lots of data from the client, the developer team has to listen carefully.
   ii. Maximum information with respect to requirement and the specifications should be collected before the implementation and the designing of the software.
2. **Prepare before you communicate**
   i. A proper agenda or the guidelines for the meetings should be prepared before the start of the meeting.
   ii. Complete detail and the description about the clients and their work area should be gathered to deliver the software up to the best expectation.
3. **Have a facilitator for any communication meeting**
   i. The requirement gathering and the specification are important for any software development, hence the communication should continue till the requirement gathering is over.
4. **Face-to-face communication is best**

i. It is always better to sit across the table and have discussion on the requirement on the software development by the client and the developer.

5. Take notes and document decisions

i. The important points discussed should also be recorded.

ii. Proper notes and the documentation is important for the successful completion and deployment of the project.

6. Strive for collaboration

i. Collaboration in terms of teamwork is required for the successful completion of the software.

ii. The collective knowledge of the team members should be implemented in the development.

7. Stay focused and modularize your discussion

i. As the development is the working of many team members, so the possibility of the discussion going from one topic to the other topic is quite possible.

ii. As a good software developer it is required that the discussion remains focused on the specified area.

8. Draw a picture if something is unclear

i. Drawing flowcharts, E-R diagrams and other supporting graphical representations give clarity to the discussion and the documentation.

9. Move on once you agree, move on when you can't agree, move on if something unclear can't be clarified at the moment

i. Healthy discussion leads to the final conclusion of successful implementation of the software.

ii. Once reached to final statement recorded should move to the next step.

iii. If no conclusion is reached than that point should be left and move ahead with new implementation which is cost effective.

10. Negotiation is not a contest or game

i. Negotiation should be mutual not to put someone down or make them feel to be the loser.

(c) Briefly describe the principles of coding.

(*Description the principles- *4 Marks)

Ans: The principles which guide the coding tasks are programming languages, programming styles and programming methods

i. Preparation principles: Before you write one line of code, be sure you

- Understand of the problem you are trying to solve
- Understand basic design, principles & concepts.
- Pick a programming language that meets the needs of the software to be build and the environment in which the software will operate.
- Select a programming environment that provides tools that will make you work simpler and easier.
- Create a set of units that will be applied once the component you code is completed.

ii. Coding principles: As you begin writing code, be sure you:

- Constrain your algorithms by following structured programming practice.
- Consider the use of pair programming.
- Select data structures that will meet the needs of the design.
- Understand the software architecture and create interfaces that are consistent with it.
- Keep conditional logic as simple as possible.
- Create nested loops in a way that makes them easily testable.
- Select meaningful variable names and follow other local coding standards.
- Write code that is self-documenting.
- Create a visual layout that aids understanding.

iii. Validation Principles: After you have completed your first coding pass, be sure you:

- Conduct a code walkthrough when appropriate.
- Perform unit tests and correct errors when you have uncovered.
- Refactor the code.

(d) With neat diagram explain analysis model.
(Diagram - 1 Mark, any three points - 3 Marks)

Ans:

(a) The model should focus on requirements that are visible within the problem or business domain and be written as a relatively high level of abstraction.

(b) Each element of the analysis model should add to the understanding of the requirements and provide insight into the information domain, function, and behaviour of the system.
(c) Delay consideration of infrastructure and other non-functional models until design.
(d) Minimize coupling throughout the system.
(e) Be certain the analysis model provides value to all stakeholders.
(f) Keep the model as simple as possible.
(g) The Figure 2 shows the structure of analysis modelling.

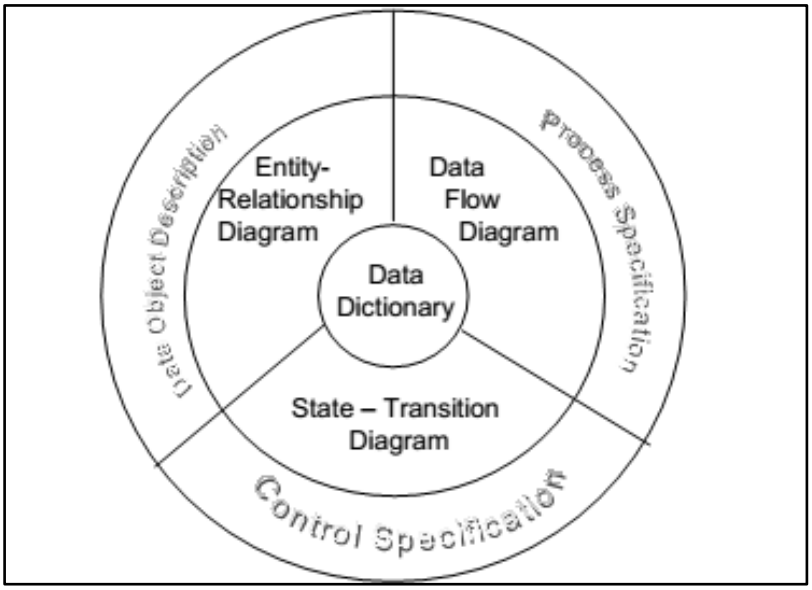


Figure Analysis Modelling

(e) Explain domain analysis with a neat diagram
(Diagram - 1 Mark, Explanation - 3 Marks)

Ans:

1. Definition

(a) Domain Analysis is the process that identifies the relevant objects of an application domain.
(b) The goal of Domain Analysis is Software Reuse.

The higher is the level of the life-cycle object to reuse, the larger are the benefits coming from its reuse, and the harder is the definition of a workable process.

2. Concept and technical application domain of the software

(a) Frameworks are excellent candidates for Domain Analysis: they are at a higher level than code but average programmers can understand them.

(b) Umbrella activity involving the Identification, analysis, and specification of common requirements from a specific application domain, typically for reuse in multiple projects

(c) Object-oriented domain analysis involves the identification, analysis, and specification of reusable capabilities within a specific application domain in terms of common objects, classes, subassemblies, and frameworks

3. Input and Output Structure of domain analysis

(a) Figure 5 shows the flow of the input and the output data in the domain analysis module.

(b) The main goal is to create the analysis classes and common functions.

(c) The input consists of the knowledge domain.

(d) The input is based on the technical survey, customer survey and expert advice.

(e) The output domain consists of using the input as the reference and developing the functional models

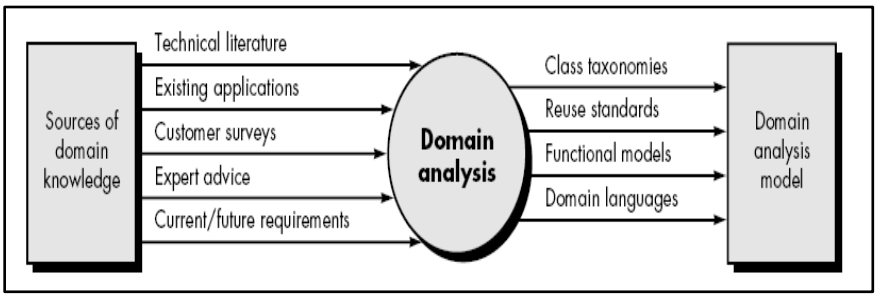


Figure: Domain Analysis

(f) Explain unit testing.

(Diagram - 1 Mark, Explanation - 3 Marks)

Ans:

(a) **Unit Testing** is a level of the software testing process where individual units/components of a software/system are tested.
(b) The purpose is to validate that each unit of the software performs as designed.

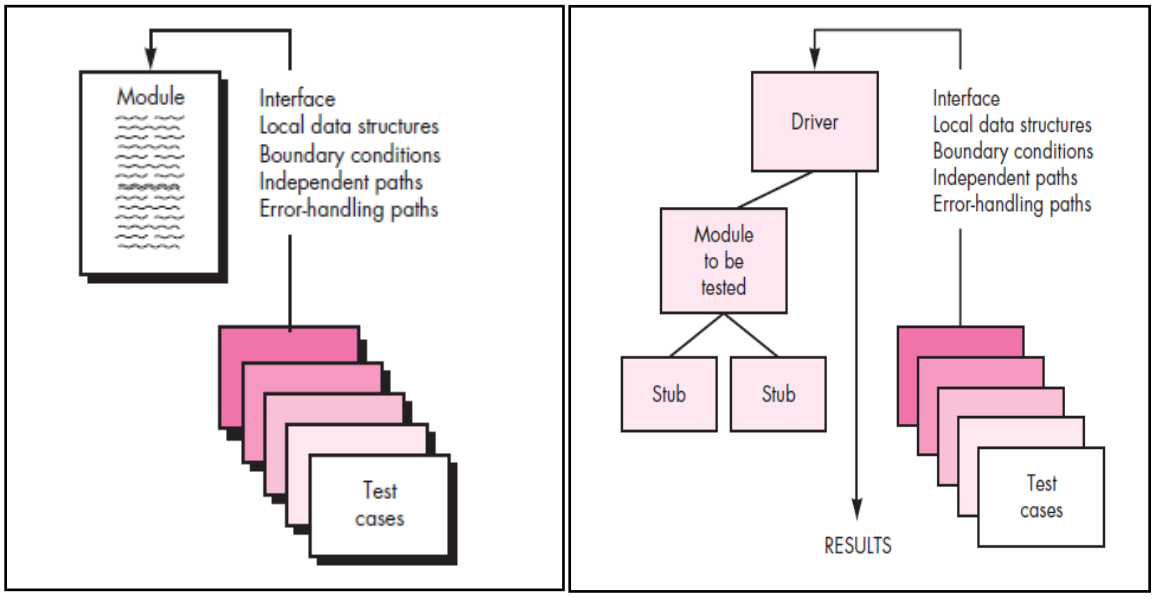


Figure: Unit Testing

(c) A unit is the smallest testable part of software.
(d) It usually has one or a few inputs and usually a single output.
(e) In procedural programming a unit may be an individual program, function, procedure, etc.
(f) In object-oriented programming, the smallest unit is a method, which may belong to a base/super class, abstract class or derived/child class.

Advantages

(a) Unit testing increases confidence in changing/maintaining code.

(b) If good unit tests are written and if they are run every time any code is changed, the likelihood of any defects due to the change being promptly caught is very high.

(c) If unit testing is not in place, the most one can do is hope for the best and wait till the test results at higher levels of testing are out.

(d) If codes are already made less interdependent to make unit testing possible, the unintended impact of changes to any code is less.

(e) Codes are more reusable. In order to make unit testing possible, codes need to be modular. This means that codes are easier to reuse.

(f) The cost of fixing a defect detected during unit testing is lesser in comparison to that of defects detected at higher levels.

(g) Compare the cost (time, effort, destruction, humiliation) of a defect detected during acceptance testing or say when the software is live.

(h) Debugging is easy. When a test fails, only the latest changes need to be debugged. With testing at higher levels, changes made over the span of several days/weeks/months need to be debugged.

Alpha Testing	Beta Testing

4. (A)Answer any THREE of the following: **12**

(a) Differentiate between alpha-testing and beta-testing.

(1 Mark each; any four points)

Ans:

Performed at Developer's site.	Performed at End user's site.
Performed in controlled Environment as developer is present.	Performed in uncontrolled environment as developer is not present.
Less probability of finding of errors as it is driven by developer.	High probability of finding errors as end user can use it the way he wants.
It is done during implementation phase of software	It is done as pre-release of software.
It is not considered as live application	It is considered as live application.
Less time consuming as developer can make necessary changes in given time.	More time consuming. As user has to report bugs if any via appropriate channel

(b) Describe the following debugging strategies:

(i) Brute force

(ii) Back tracking

(2 Marks Each)

Ans:

i. Brute force

a. This method is most common and least efficient for isolating the cause of a software error.

b. We apply this method when all else fail. In this method, a printout of all registers and relevant memory locations is obtained and studied.

c. All dumps should be well documented and retained for possible use on subsequent problems.

ii. Back tracking

a. It is a quite popular approach of debugging which is used effectively in case of small applications.

b. The process starts from the site where a particular symptom gets detected, from there on backward tracing is done across the entire source code till we are able to lay our hands on the site being the cause.

c. As the number of source lines increases, the number of potential backward paths may become unmanageably large.

(c) With an example, explain how CPM & PERT are useful in software project management.

(2 Marks Each, disadvantages optional)

Ans:

Program Evaluation Review Technique (PERT)

i. PERT, is used in projects that have unpredictable tasks and activities such as in research and development projects.
ii. It utilizes three estimates of the time to complete the project: the most probable, the most promising, and the most unfavorable.
iii. It is a probabilistic tool using several estimates to determine the time completion of the project and to control the activities involved in the project so that it will be completed faster and at a lower cost.

Critical Path Method (CPM)

i. CPM is a technique that is used in projects that have predictable activities and tasks such as in construction projects.
ii. It allows project planners to decide which aspect of the project to reduce or increase when a trade-off is needed.
iii. It is a deterministic tool and provides an estimate on the cost and the amount of time to spend in order to complete the project.
iv. It allows planners to control both the time and cost of the project.

Disadvantage of PERT and CPM

i. The Program Evaluation and Review Technique (PERT) is a project management technique or tool which is suitable for projects that have unpredictable activities while the Critical Path Method (CPM) is a project management tool which is suitable for projects that have predictable activities.

ii. CPM uses a single estimate for the time that a project can be completed while PERT uses three estimates for the time that it can be completed.

iii. CPM is a deterministic project management tool while PERT is a probabilistic project management tool.

iv. CPM allows project management planners to determine which aspect of the project to sacrifice when a trade-off is needed in order to complete the project while PERT does not.

.

(d) Give the outline that defines basic elements of ISO 9001:2000 for software quality assurance.

(Diagram Optional, Any RelevantDescription- 4 Marks)

Ans:

i. International set of standards for quality management

ii. Quality standards and procedures must be documented in an organizational quality manual

iii. An external body is often used to certify that the quality manual conforms to ISO 9000 standards

iv. Many customers are demanding that suppliers are ISO 9000 certified

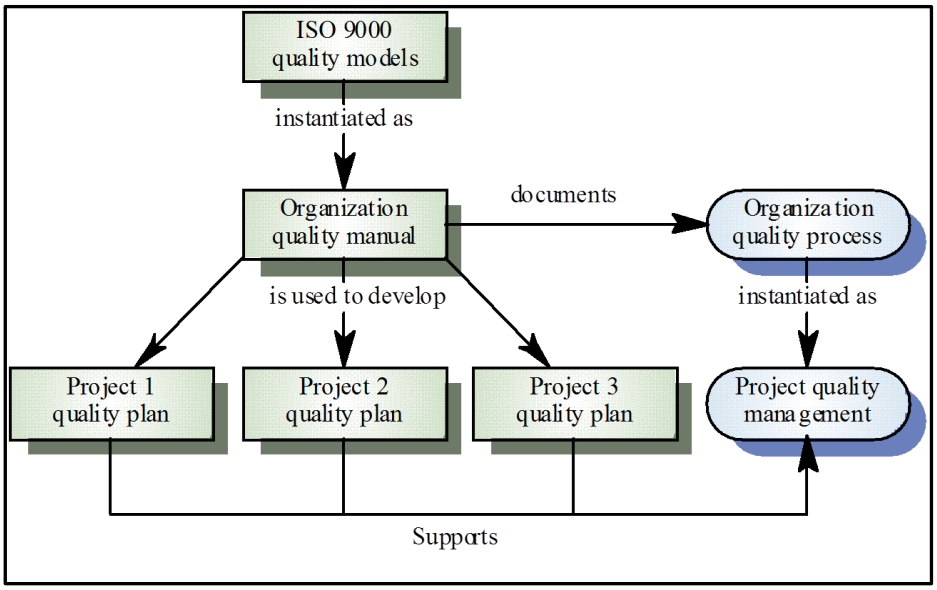


Figure: ISO Model

(B) Answer any ONE: **6**

(a) Explain the basic principles of project scheduling.

(Any 6principles of project scheduling - 1 Mark each)

Ans:

i. Compartmentalization: The project must be compartmentalized into a number of manageable activities and tasks.
ii. Interdependency: The interdependency of each compartmentalized activity or task must be determined.
iii. Time allocation: Each task to be scheduled must be allocated some number of work units (e.g., person-days of effort).
iv. Effort validation: the project manager must ensure that no more than the allocated number of people have been scheduled at any given time.
v. Defined responsibilities: Every task that is scheduled should be assigned to a specific team member
vi. Defined outcomes: Every task that is scheduled should have a defined outcome.
vii. Defined milestones: Every task or group of tasks should be associated with a project milestone.
viii. A milestone is accomplished when one or more work products has been reviewed for quality and has been approved.

(b) Explain the McCall's Quality factors.

((Diagram - 2 Marks, Any Four factors - 1 Mark each)

Ans:

The factors that affect S/W quality can be categorized in two broad groups:

1. Factors that can be directly measured (defects uncovered during testing)
2. Factors that can be measured only indirectly (Usability and maintainability)

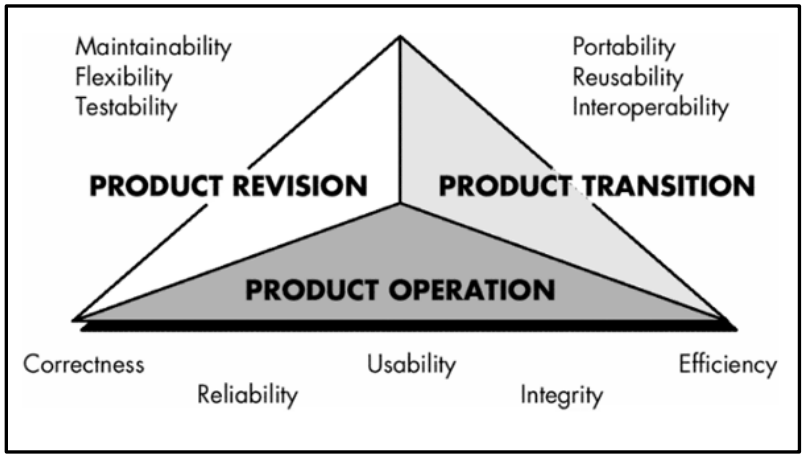


Figure: McCall's Quality Factor

The S/W quality factors shown above focus on three important aspects of a S/W product:

i. Its operational characteristics
ii. Its ability to undergo change
iii. Its adaptability to new environments

The various factors of quality are:

(a) Correctness*:* The extent to which a program satisfies its specs and fulfils the customer's mission objectives.

(b) Reliability: The extent to which a program can be expected to perform its intended function with required precision.

(c) Efficiency: The amount of computing resources and code required to perform is function.

(d) Integrity: The extent to which access to S/W or data by unauthorized persons can be controlled.

(e) Usability: The effort required to learn, operate, prepare input for, and interpret output of a program.

(f) Maintainability: The effort required to locate and fix errors in a program.

(g) Flexibility: The effort required to modify an operational program.

(h) Testability: The effort required to test a program to ensure that it performs its intended function.

(i) Portability: The effort required to transfer the program from one hardware and/or software system environment to another.

(j) Reusability: The extent to which a program can be reused in other applications-related to the packaging and scope f the functions that the program performs.

(k) Interoperability: The effort required to couple one system to another.

5. Answer any TWO of the following: **16**

(a) Explain the core principles of software engineering in details
(Each principle – 1 Mark; list of all principles – 1 Mark)

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) Explain in details RMMM strategy.
(Risk Mitigation – 3 Marks; Risk Monitoring – 3 Marks; Risk Management – 2 Marks)

Ans:

To assist the project team in developing a strategy for dealing with risk. An effective strategy must consider three issues: risk avoidance, risk monitoring, and risk management and contingency planning. If a software team adopts a proactive approach to risk, avoidance is always the best strategy. This is achieved by developing a plan for risk mitigation.

Risk Mitigation: -

To mitigate this risk, you would develop a strategy for reducing turnover. Among the possible steps to be taken are:

- Meet with current staff to determine causes for turnover (e.g., poor working conditions, low pay, competitive job market).
- Mitigate those causes that are under your control before the project starts.
- Once the project commences, assume turnover will occur and develop techniques to ensure continuity when people leave.
- Organize project teams so that information about each development activity is widely dispersed.
- Define work product standards and establish mechanisms to be sure that all models and documents are developed in a timely manner.
- Conduct peer reviews of all work (so that more than one person is “up to speed”).
- Assign a backup staff member for every critical technologist.

Risk Monitoring: -

As the project proceeds, risk-monitoring activities commence. The project manager monitors factors that may provide an indication of whether the risk is becoming more or

less likely. In the case of high staff turnover, the general attitude of team members based on project pressures, the degree to which the team has jelled, interpersonal relationships among team members, potential problems with compensation and benefits, and the availability of jobs within the company and outside it are all monitored.

Risk Management: -

In addition to monitoring these factors, a project manager should monitor the effectiveness of risk mitigation steps. Risk management and contingency planning assumes that mitigation efforts have failed and that the risk has become a reality.

Risk management and contingency planning assumes that mitigation efforts have failed and that the risk has become a reality.

It is important to note that risk mitigation, monitoring, and management (RMMM) steps incur additional project cost. For example, spending the time to back-up every critical technologist costs money. Part of risk management, therefore, is to evaluate when the benefits accrued by the RMMM steps are outweighed by the costs associated with implementing them.

(c) What is six sigma? Describe the core steps of DMAIC in details.
(Six Sigma – 4 Marks; DMAIC – 4 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:

DMAIC: -

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

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,

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.

6. Answer any FOUR of the following: **16**

(a) Explain white box testing.
(White box testing - 2 Marks; use of White box - 2 Marks)

Ans:

White-box testing, sometimes called glass-box testing is a test-case design philosophy that uses the control structure described as part of component-level design to derive test cases. Using white-box testing methods, you can derive test cases that;

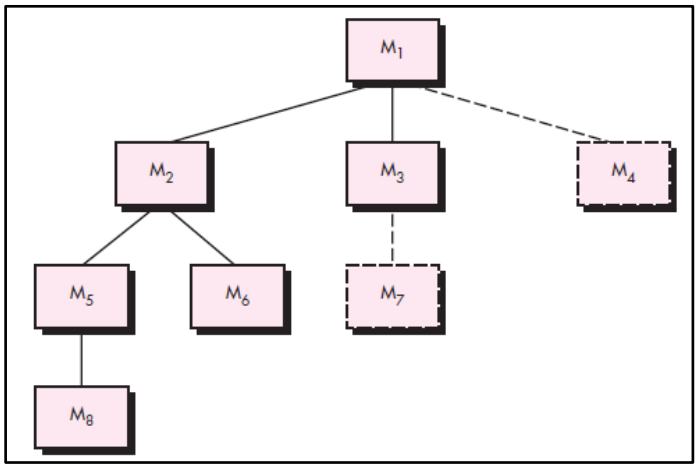
Use of White box testing:

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

(b) Explain Top-Down integration testing
(Description – 3 Marks; Diagram – 1 Mark)

Ans:

Top-down integration: Top-down integration testing is an incremental approach to construction of the software architecture. Modules are integrated by moving downward through the control hierarchy, beginning with the main control module (main program). Modules subordinate (and ultimately subordinate) to the main control module are incorporated into the structure in either a depth-first or breadth-first manner.



The integration process is performed in a series of five steps:

1. The main control module is used as a test driver and stubs are substituted for all components directly subordinate to the main control module.
2. Depending on the integration approach selected (i.e., depth or breadth first), subordinate stubs are replaced one at a time with actual components.
3. Tests are conducted as each component is integrated.
4. On completion of each set of tests, another stub is replaced with the real component.
5. Regression testing (discussed later in this section) may be conducted to ensure that new errors have not been introduced.

The top-down integration strategy verifies major control or decision points early in the test process. Top-down strategy sounds relatively uncomplicated, but in practice, logistical problems can arise. The most common of these problems occurs when processing at low levels in the hierarchy is required to adequately test upper levels.

(c) Describe the attributes of a good test.
(Description of Each attribute - 1 Mark)

Ans:

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.

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).

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.

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) Why do the software projects fail? Give reasons.
(Each reason - 1 Mark; any four reasons)

Ans:

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

- An unrealistic deadline established by someone outside the software team and forced on managers and practitioners.
- Changing customer requirements that are not reflected in schedule changes.
- An honest underestimate of the amount of effort and/or the number of resources 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.

(e) Describe the four elements of software configuration management system.
(*Describe*of Each attribute - 1 Mark)

Ans:

Four important elements that should exist when a configuration management system is developed:

- **Component elements -**a set of tools coupled within a file management system (e.g., a database) that enables access to and management of each software configuration item.
- **Process elements -**a collection of actions and tasks that define an effective approach to change management (and related activities) for all constituencies involved in the management, engineering, and use of computer software.
- **Construction elements** -a set of tools that automate the construction of software by ensuring that the proper set of validated components (i.e., the correct version) have been assembled.
- **Human elements**-a set of tools and process features (encompassing other CM elements) used by the software team to implement effective SCM.