

Q.1 Attempt any **FIVE** of the following : **[20]**

Q.1(a) Explain changing nature of software. **[4]**

Ans.: **Changing Nature of Software** : Whenever one starts with the software implementation changes can occur any time. The software can be change due to any reason. But while implementing software one should be ready for such changes as if changes occur there shall not be drastic change in the system. The development team should manage to implement/mould the implemented system so that the changes can be reflected and the user requirements meet. When change occur the team look for the current status of the system and from there onwards they starts implementing a system with new requirements of a user or changes which is to be implemented in a system.

Q.1(b) What are communication principles? Explain their meaning. **[4]**

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, do some research to understand business domain. 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, to mediate any conflict that does occur, and to ensure that other principles are followed.

Principle 4. Face-to-face communication is best. Face to face communication is always makes sense. It usually works better when some other representation of the relevant information is present. For example, a participant may create a drawing document that serves as a focus for discussion.

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 occurs when the collective knowledge of members of the team is used to describe product or system functions or features. Each small collaboration serves to build trust among team members and creates a common goal for the team.

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 conversations 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.

Q.1(c) List four objectives of testing.

[4]

Ans.: List of 4 objectives of testing

1. Testing is a process of executing a program with the intent of finding an error.
2. A good test case is one that has a high probability of finding an as-yet undiscovered error.
3. A successful test is one that uncovers an as-yet-undiscovered error
4. All tests should be traceable to customer requirements.
5. A good test has a high probability of finding an error.
6. A good test is not redundant.
7. A good test should be –best of breed.
8. A good test should be neither too simple nor too complex.

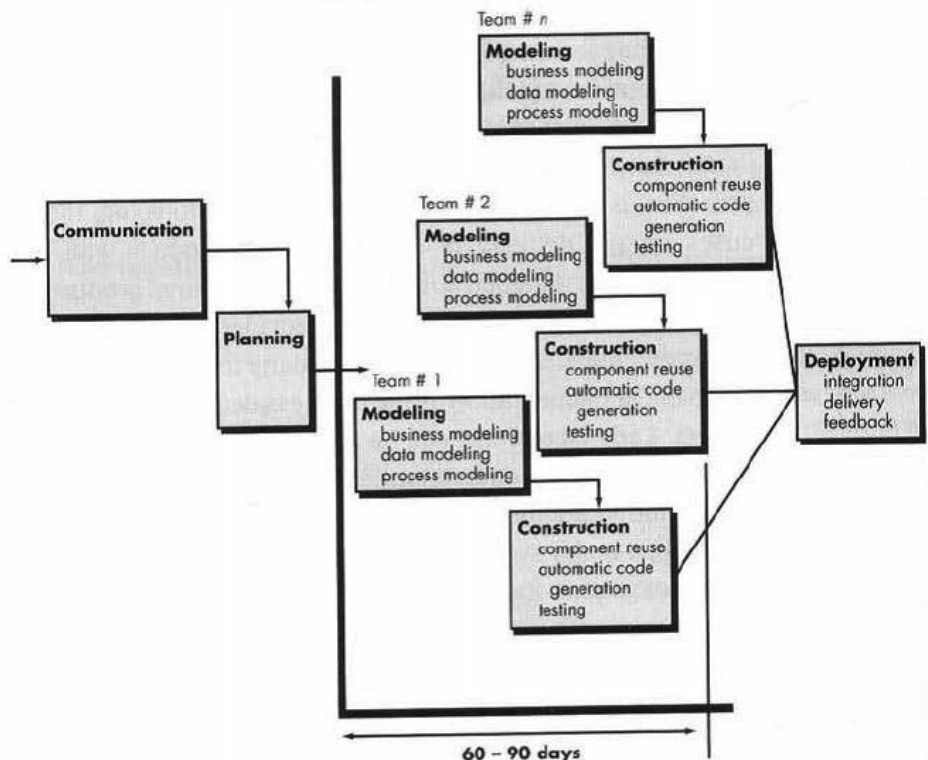
OR

1. Finding programming defects.
2. Gaining confidence in and providing information about the level of quality.
3. To make sure that the end result meets the business and user requirements.
4. To ensure that it satisfies SRS that is System Requirement Specifications.

Q.1(d) Explain RAD model with its advantages and disadvantages.

[4]

Ans.: The RAD Model:



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.

This approach is very useful if the business application requirements are modularized as function to be completed by individual teams and finally to integrate into a complete system. As such compared to waterfall model the team will be of larger size to function with proper coordination.

RAD model distributes the analysis and construction phases into a series of short iterative development cycles. The activities of each phase per team are Business modeling, Data modeling and process modeling.

This model is useful for projects with possibility of modularization.

RAD may fail if modularization is difficult. This model should be used if domain experts are available with relevant business knowledge.

Advantages:

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

Disadvantages:

1. Success of RAD model depends on strong technical team expertise and skills.
2. Highly skilled developers needed with modeling skills.
3. User involvement throughout life cycle. If developers & customers are not committed to the rapid fire activities necessary to complete the System in a much-abbreviated time frame, RAD projects will fail.
4. May not be appropriate for very large scale systems where the technical risks are high.

Q.1(e) What is alpha-beta testing?

[4]

Ans.: **Alpha Testing:** The *alpha test* is conducted at the developer's site by a customer. The software is used in a natural setting with the developer "looking over the shoulder" of the user and recording errors and usage problems. Alpha tests are conducted in a controlled environment.

Beta Testing: The *beta test* is conducted at one or more customer sites by the end-user of the software. Unlike alpha testing, the developer is generally not present. Therefore, the beta test is a "live" application of the software in an environment that cannot be controlled by the developer. The customer records all problems (real or imagined) that are encountered during beta testing and reports these to the developer at regular intervals.

As a result of problems reported during beta tests, software engineers make modifications and then prepare for release of the software product to the entire customer base.

Q.1(f) Describe six sigma for software engineering. [4]

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:

- 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.
- 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.

Q.1(g) Explain analysis modeling. [4]

Ans.: The analysis model and requirements specification provide a means for assessing quality once the software is built. Requirements analysis results in the specification of software's operational characteristics.

The analysis model is a bridge between the system description and the design model.

Objectives Analysis model must achieve three primary objectives: Describe Customer needs
Establish a basis for software design Define a set of requirements that can be validated once the software is built.

Analysis Rules of Thumb

- The model should focus on requirements that are visible within the problem or business domain. The level of abstraction should be relatively high.
- Each element of the analysis model should add to overall understanding of software requirements and provide insight into the information, function, and behavior domains of the system.
- Delay consideration of infrastructure and other non-functional models until design.
- For example, a database may be required, but the classes necessary to implement it, the functions required to access it, and the behavior that will be exhibited as it is used should be considered only after problem domain analysis has been completed.
- Minimize coupling throughout the system.

- The level of interconnectedness between classes and functions should be reduced to a minimum.
- Be certain that the analysis model provides value to all stakeholders.
- Each constituent has its own use for the model.
- Keep the model as simple as it can be.
- Ex: Don't add additional diagrams when they provide no new information.
- Only modeling elements that have values should be implemented.

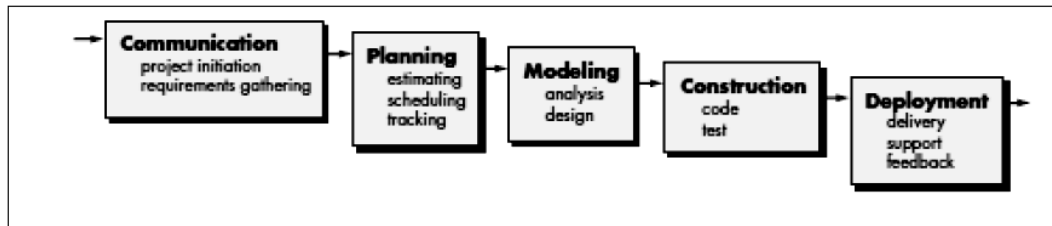
Q.2 Attempt any FOUR of the following :

[16]

Q.2(a) Explain the waterfall model.

[4]

Ans.:



The waterfall model is a traditional method, sometimes called the classic life cycle. This is one of the initial models. As the figure implies stages are cascaded and shall be developed one after the other. It suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through, *communication, planning, modeling construction and deployment.*

In other words one stage should be completed before the other begins. Hence, when all the requirements are elicited by the customer, analyzed for completeness and consistency, documented as per requirements, the development and design activities commence. One of the main needs of this model is the user's explicit prescription of complete requirements at the start of development. For developers it is useful to layout what they need to do at the initial stages. Its simplicity makes it easy to explain to customers who may not be aware of software development process. It makes explicit with intermediate products to begin at every stage of development. One of the biggest limitation is it does not reflect the way code is really developed. Problem is well understood but software is developed with great deal of iteration. Often this is a solution to a problem which was not solved earlier and hence software developers shall have extensive experience to develop such application; as neither the user nor the developers are aware of the key factors affecting the desired outcome and the time needed. Hence at times the software development process may remain uncontrolled. Today software work is fast paced and subject to a never-ending stream of changes in features, functions and information content. Waterfall model is inappropriate for such work. This model is useful in situation where the requirements are fixed and work proceeds to completion in a linear manner.

Q.2(b) Explain modeling practice in software engineering with principles.

[4]

Ans.: We create models to gain a better understanding of the actual entity to be built.

- **Principle 1. The primary goal of the software team is to build software, not create models.**

Agility means getting software to the customer in the fastest possible time. Models that make this happen are worth creating, but models that slow the process down or provide little new insight should be avoided.

- **Principle 2. Travel light—don't create more models than you need.**

Every model that is created must be kept up-to-date as changes occur. More importantly, every new model takes time that might otherwise be spent on construction (coding and testing). Therefore, create only those models that make it easier and faster to construct the software.

- **Principle 3. Strive to produce the simplest model that will describe the problem or the software.**
- Don't overbuild the software by keeping models simple, the resultant software will also be simple. The result is software that is easier to integrate, easier to test, and easier to maintain (to change). In addition, simple models are easier for members of the software team to understand and critique, resulting in an ongoing form of feedback that optimizes the end result.
- **Principle 4. Build models in a way that makes them amenable to change.**
Assume that your models will change, but in making this assumption don't get sloppy. For example, since requirements will change, there is a tendency to give requirements models short. Why? Because you know that they'll change anyway.
The problem with this attitude is that without a reasonably complete requirements model, you'll create a design (design model) that will invariably miss important functions and features.
- **Principle 5. Be able to state an explicit purpose for each model that is created.**
Every time you create a model, ask yourself why you're doing so. If you can't provide solid justification for the existence of the model, don't spend time on it.
- **Principle 6. Adapt the models you develop to the system at hand.**
It may be necessary to adapt model notation or rules to the application; for example, a video game application might require a different modeling technique than real-time, embedded software that controls an automobile engine.
- **Principle 7. Try to build useful models, but forget about building perfect models.**
When building requirements and design models, a software engineer reaches a point of diminishing returns. That is, the effort required to make the model absolutely complete and internally consistent is not worth the benefits of these properties. Am I suggesting that modeling should be sloppy or low quality? The answer is "no." But modeling should be conducted with an eye to the next software engineering steps. Iterating endlessly to make a model "perfect" does not serve the need for agility.
- **Principle 8. Don't become dogmatic about the syntax of the model.**
If it communicates content successfully, representation is secondary. Although everyone on a software team should try to use consistent notation during modeling, the most important characteristic of the model is to communicate information that enables the next software engineering task. If a model does this successfully, incorrect syntax can be forgiven.
- **Principle 9. If your instincts tell you a model isn't right even though it seems okay on paper, you probably have reason to be concerned.** If you are an experienced software engineer, trust your instincts. Software work teaches many lessons—some of them on a subconscious level. If something tells you that a design model is doomed to fail (even though you can't prove it explicitly), you have reason to spend additional time examining the model or developing a different one.
- **Principle 10. Get feedback as soon as you can.**
Every model should be reviewed by members of the software team. The intent of these reviews is to provide feedback that can be used to correct modeling mistakes, change misinterpretations, and add features or functions that were inadvertently omitted.

Q.2(c) Explain principle of scheduling.

[4]

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 is a function of the interdependencies.

Start and stop dates are also established based on whether work will be conducted on a full-time or part-time basis.

Effort validation: Every project has a defined number of people on the team. As time allocation occurs, the project manager must ensure that no more than the allocated number of people has been scheduled at any given time.

Defined responsibilities: Every task that is scheduled should be assigned to a specific team member.

Defined outcomes: Every task that is scheduled should have a defined outcome for software projects such as a work product or part of a work product. Work products are often combined in deliverables.

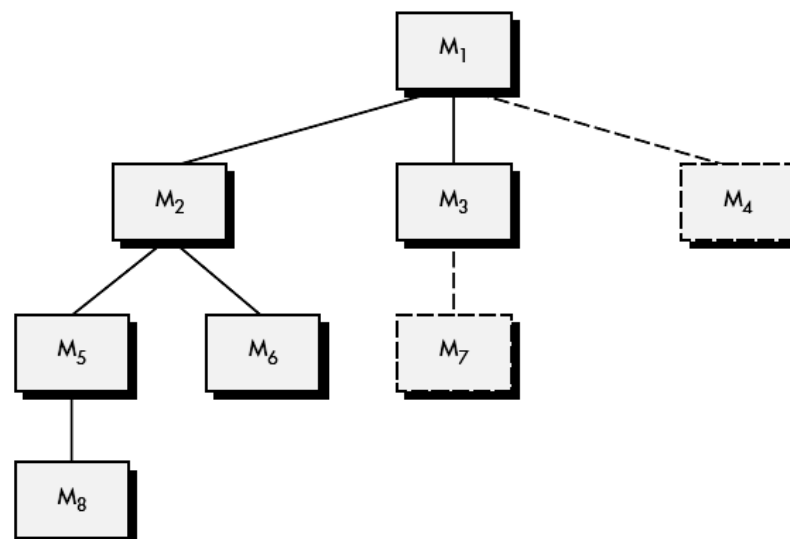
Defined milestones: Every task or group of tasks should be associated with a project milestone. A milestone is accomplished when one or more work products has been reviewed for quality and has been approved.

Q.2(d) Describe integration testing approach.

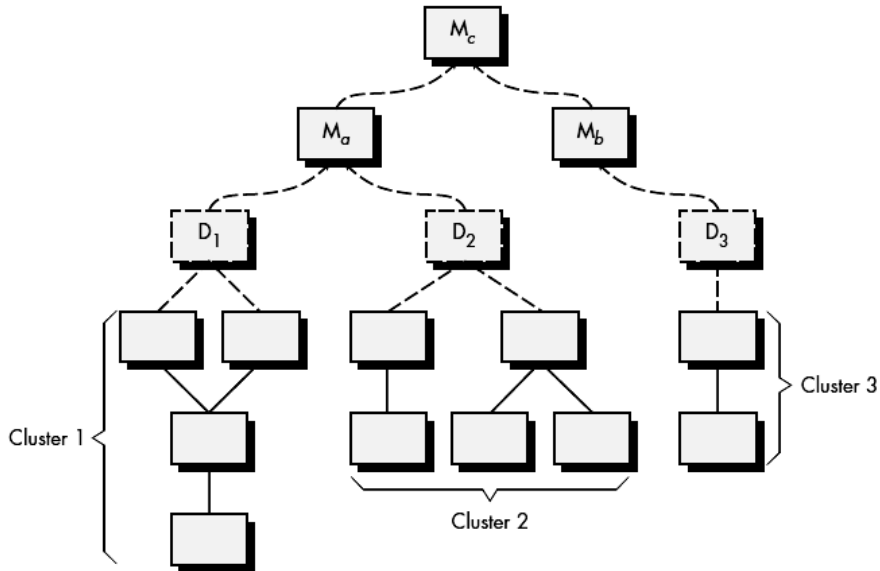
[4]

Ans.: Integration testing is a systematic technique for constructing the software architecture while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit-tested components and build a program structure that has been dictated by design. There are two approaches used in Integration Testing as follows:

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. Modules subordinate (and ultimately subordinate) to the main control module are incorporated into the structure in either a depth-first or breadth-first manner.



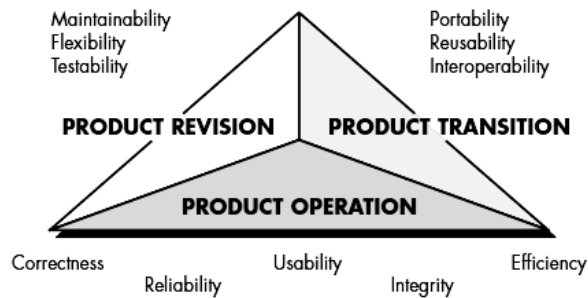
Bottom-up integration. Bottom-up integration testing, as its name implies, begins construction and testing with atomic modules (i.e., components at the lowest levels in the program structure). Because components are integrated from the bottom up, the functionality provided by components subordinate to a given level is always available and the need for stubs is eliminated.



Q.2(e) Explain Mccalls quality factor.

[4]

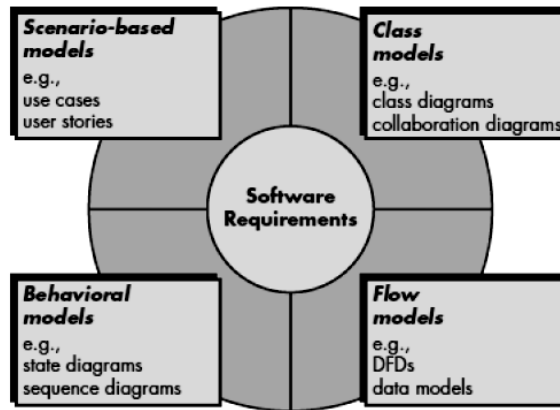
Ans. :



- **Correctness.** The extent to which a program satisfies its specification and fulfills the customer's mission objectives.
- **Reliability.** The extent to which a program can be expected to perform its intended function with required precision.
- **Efficiency.** The amount of computing resources and code required by a program to perform its function.
- **Integrity.** Extent to which access to software or data by unauthorized persons can be controlled of a program
- **Maintainability.** Effort required to locate and fix an error in a program.
- **Flexibility.** Effort required to modify an operational program.
- **Testability.** Effort required to test a program to ensure that it performs its intended function.
- **Portability.** Effort required to transfer the program from one hardware and/or software system environment to another.
- **Reusability.** Extent to which a program [or parts of a program] can be reused in other applications—related to the packaging and scope of the functions that the program performs.
- **Interoperability.** Effort required to couple one system to another.

Q.2(f) What is an object oriented analysis? [4]

Ans.: Object-oriented Analysis focuses on the definition of classes and the manner in which they collaborate with one another to effect customer requirements.



Elements of the Analysis Model

The intent is to define all classes, relationship, behavior associated with them, that are to the problem to be solved. To achieve this following task should occur.

- Task 1 : Basic user requirements must be communicated between user and developer.
- Task 2 : Classes must be identified (i.e. attributes, methods defined)
- Task 3 : A class hierarchy is defined
- Task 4 : Object- object relationships (object connection) should be represented.
- Task 5 : Object behavior must be modeled.
- Task 6 : Task-1 to Task-5 is reapplied iteratively till model is complete.

Q.3 Attempt any FOUR of the following : [16]

Q.3(a) Difference between prescriptive and agile process model. [4]

Ans.:

	Prescriptive Process Model	Agile Process Model
1.	Product Oriented process. Process and technology are crucial	People oriented process. Favors people over technology
2.	A traditional approach for software product development	It is an recent approach for Project Management
3.	Traditional and modern approaches using generic process framework activities with medium to large cycle time	Cycle-time reduction is most important
4.	Focus is on tasks, tools such as estimating, scheduling, tracking and control	Model focuses on modularity, iterative, time bound, parsimony, adaptive, incremental convergent, collaborative approach
5.	Models include Waterfall, Incremental, Prototype, RAD and spiral	Agile process model uses the concept of Extreme Programming

Q.3(b) Describe any two core principles of software engineering. [4]

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.

Q.3(c) What is test plan? [4]

Ans.: **Test plan:** A document describing the scope, approach, resources and schedule of intended test activities. It identifies amongst others test items, the features to be tested, the testing tasks, who will do each task, degree of tester independence, the test environment, the test design techniques and entry and exit criteria to be used, and the rationale for their choice, and any risks requiring contingency planning. It is a record of the test planning process
master test plan: A test plan that typically addresses multiple test levels.
phase test plan: A test plan that typically addresses one test phase.

Q.3(d) Differentiate between Black Box Testing and White Box Testing. [4]

Ans.:

	Black Box Testing	White Box Testing
Definition	Black Box Testing is a software testing method in which the internal structure/design/ implementation of the item being tested is NOT known to the tester	White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester.
Levels Applicable To	Mainly applicable to higher levels of testing: Acceptance Testing System Testing	Mainly applicable to lower levels of testing: Unit Testing Integration Testing
Responsibility	Generally independent Software Testers	Generally Software Developers
Programming Knowledge	Not Required	Required
Implementation Knowledge	Not Required	Required
Basis for Test Cases	Requirement Specifications	Detail Design
Diagram		

Q.3(e) Explain concept of data modeling with [4]

- (i) data object
- (ii) Data Attributes
- (iii) Cardinality
- (iv) Modality

Ans.: (i) **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.

(ii) 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".

(iii) 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);

(iv) Modality

- 1) A modality of relationship is zero if occurrence of relationship is optional and modality of relationship is 1 if occurrence of relationship is mandatory (i.e. compulsory).
- 2) The modality specifies the minimum number of relationship.
- 3) Shows maximum 1 to minimum or compulsory 1.
- 4) Example – exactly one (maximum 1 and minimum 1) room is occupied by zero or many (maximum many and minimum 0) employees.

Q.3(f) What is SPM? Why is it needed?

[4]

Ans.: The Software Project Management includes basic function such as scoping, planning, estimating, scheduling, organizing, directing, coordinating, controlling and closing. The effective Software Project Management focuses on the four P's via People, Product, Process and Project.

Project management software caters to the following primary functions:

1. **Project planning** : To define a project schedule, a project manager (PM) may use the software to map project tasks and visually describe task interactions.
2. **Task management** : Allows for the creation and assignment of tasks, deadlines and status reports.
3. **Document sharing and collaboration** : Productivity is increased via a central document repository accessed by project stakeholders.
4. **Calendar and contact sharing** : Project timelines include scheduled meetings, activity dates and contacts that should automatically update across all PM and stakeholder calendars.
5. **Bug and error management** : Project management software facilitates bug and error reporting, viewing, notifying and updating for stakeholders.
6. **Time tracking** : Software must have the ability to track time for all tasks maintain records for third-party consultants.

Q.4 Attempt any FOUR of the following :

[16]

Q.4(a) Explain the concept of software requirement specification.

[4]

Ans.: A software requirements specification (SRS) is a complete description of the behavior of the system to be developed. It includes a set of use cases describe all of the interactions that the users will have with the software. In addition to use cases, the SRS contains functional requirements and nonfunctional requirements. Functional requirements define the

internal workings of the software: that is, the calculations, technical details, data manipulation and processing, and other specific functionality that shows how the use cases are to be satisfied. Non-functional requirements impose constraints on the design or implementation (such as performance requirements, quality standards, or design constraints).

The purpose of SRS document is providing a detailed overview of software product, its parameters and goals. SRS document describes the project's target audience and its user interface, hardware and software requirements. It defines how client, team and audience see the product and its functionality.

The importance of standard template for SRS documents:

Establish the basis for agreement between the customers and the suppliers on what the software product is to do. The complete description of the functions to be performed by the software specified in the SRS will assist the potential users to determine if the software specified meets their needs or how the software must be modified to meet their needs. Reduce the development effort. The preparation of the SRS forces the various concerned groups in the customer's organization to consider rigorously all of the requirements before design begins and reduces later redesign, recoding, and retesting.

Careful review of the requirements in the SRS can reveal omissions, misunderstandings, and inconsistencies early i/p the development cycle when these problems are easier to correct. Provide a basis for estimating costs and schedules. The description of the product to be developed as given in the SRS is a realistic basis for estimating project costs and can be used to obtain approval for bids or price estimates.

Provide a baseline for validation and verification. Organizations can develop their validation and Verification plans much more productively from a good SRS. As a part of the development contract, the SRS provides a baseline against which compliance can be measured.

Facilitate transfer. The SRS makes it easier to transfer the software product to new users or new machines. Customers thus find it easier to transfer the software to other parts of their organization, and suppliers find it easier to transfer it to new customers.

Serve as a basis for enhancement. Because the-SRS discusses the product but not the project that developed it, the SRS serves as a basis for later enhancement of the finished product. The SRS may need to be altered, but it does provide a foundation for continued production evaluation.

Q.4(b) Explain characteristics of software testing.

[4]

- Ans.:**
1. To perform effective testing, a software team should conduct effective Formal Technical Reviews (FTRs) using which many errors are eliminated before testing
 2. Testing begins at the component level and works "outward" toward the integration of the entire computer-based system.
 3. Different testing techniques are appropriate for different software engineering approaches and at different points in time.
 4. Testing is conducted by the developer of the software and (for large projects) an independent test group.
 5. Testing and debugging are different activities, but debugging must be accommodated in any testing strategy

Q.4(c) State eight benefit of ISO standards.

[4]

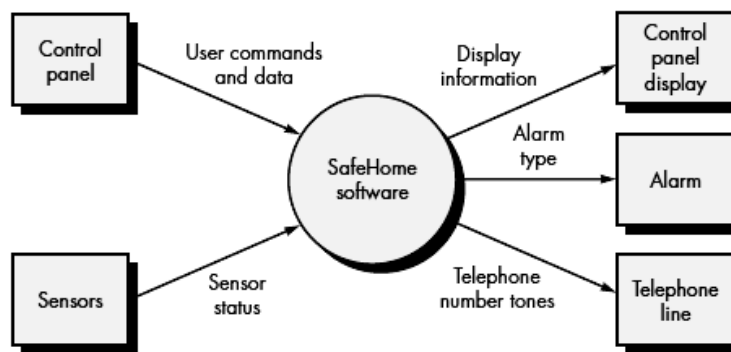
- Ans.:**
1. Well defined and documented procedures improve the consistency of output.
 2. Quality is constantly measured.
 3. Procedures ensure corrective action is taken whenever defects occur.
 4. Defect rates decrease.
 5. Defects are caught earlier and are corrected at a lower cost.
 6. Defining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

Q.4(d) Explain DFD with example.

[4]

Ans.: Data Flow Diagram (DFD) is a graphical representation of how data is actually flowing within system. It gives clear idea about which module requires what data as an input and what will be the output of that module. Generally DFD has several levels; higher the level better understanding about the system can achieve. First level of DFD is known as Context level or Level 0 which gives overall working of System. Level 1 gives modularize representation of system containing primary modules of system. From Level 2 onwards a designer starts revisiting each and every module to go in depth analysis of system which contains smaller functions to be performed by every module.

Example:



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.

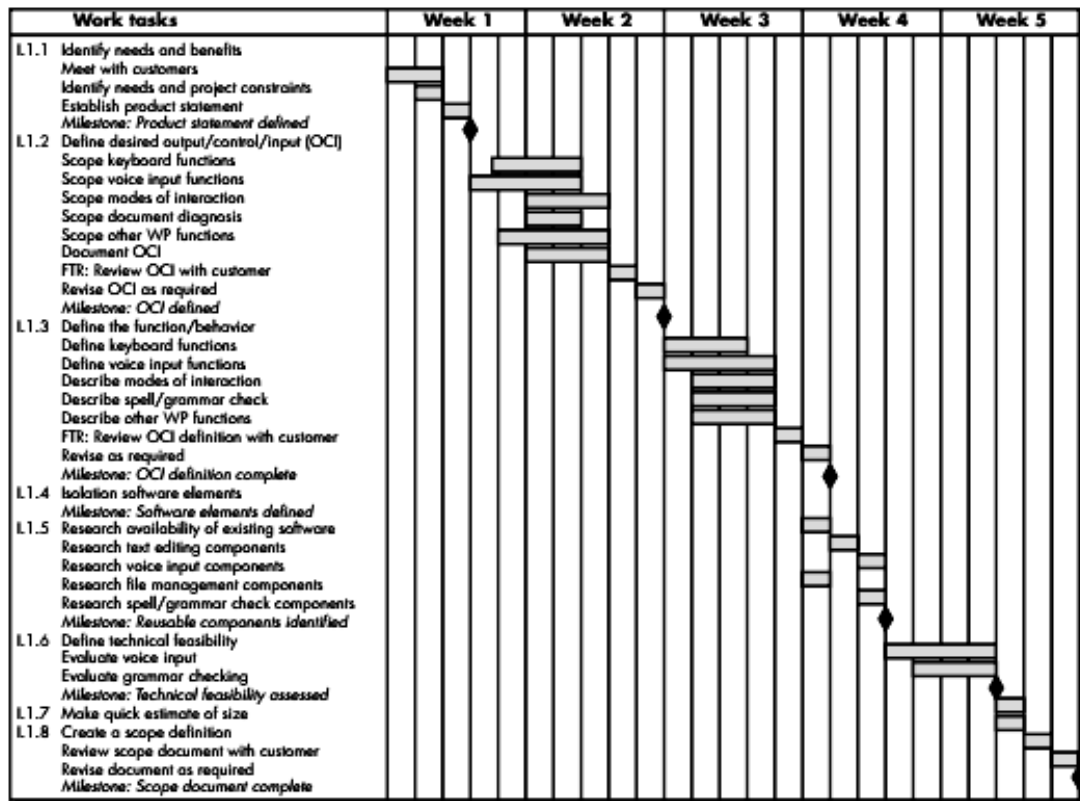
Q.4(e) Explain the concept of Gantt chart.

[4]

Ans.: When creating a software project schedule, software team begins with a set of tasks (the work breakdown structure). If automated tools are used, the work breakdown is input as a task network or task outline. Effort, duration, and start date are then input for each task. In addition, tasks may be assigned to specific individuals.

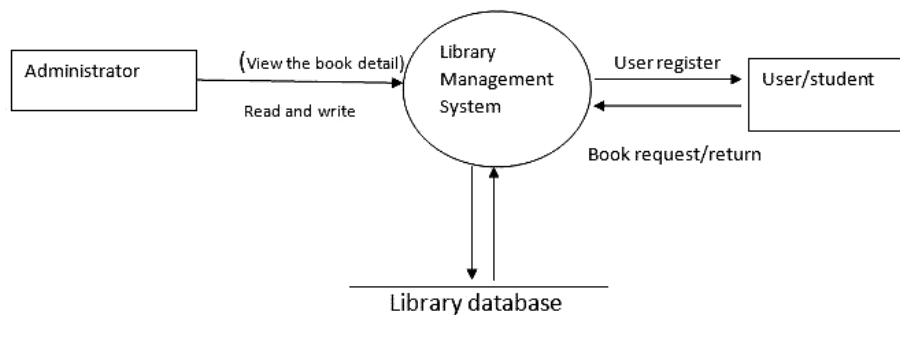
As a consequence of this input, a Gantt chart, also called a time-line chart, is generated. A Gantt chart can be developed for the entire project. Alternatively, separate charts can be developed for each project function or for each individual working on the project.

Figure below illustrates the format of a Gantt chart. It depicts a part of a software project schedule that emphasizes the concept scoping task for a word-processing (WP) software product. All project tasks are listed in the left-hand column. The horizontal bars indicate the duration of each task. When multiple bars occur at the same time on the calendar, task concurrency is implied. The diamonds indicate milestones.

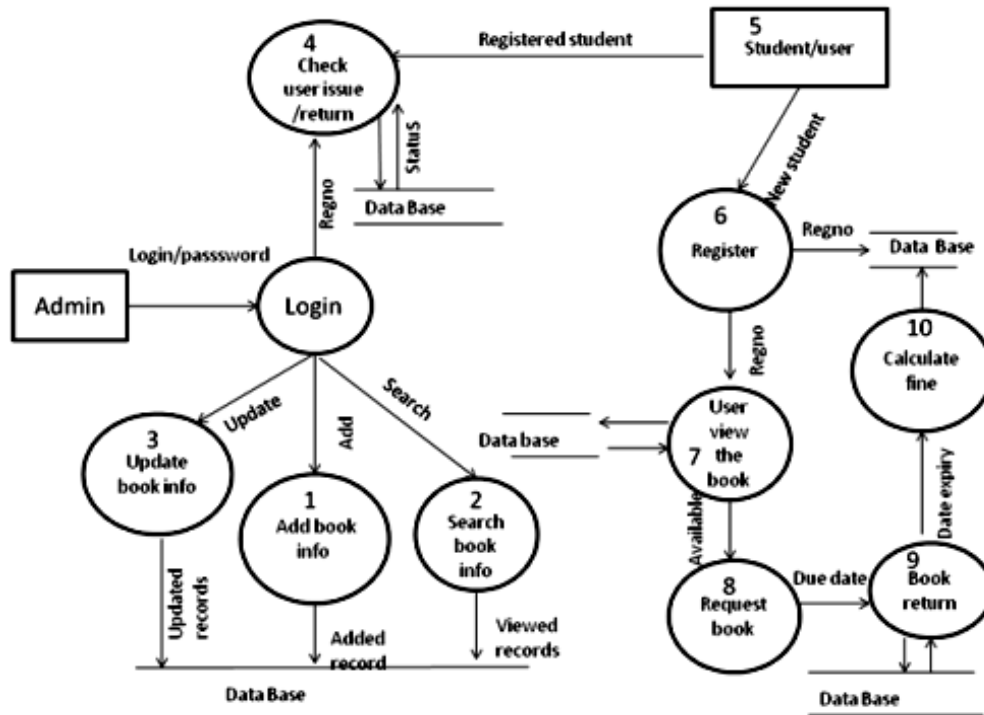


Q.4(f) For library management system draw level 0 and level 1 DFD. [4]

Ans. :



DFD Level 0 for Library Management System



DFD Level 1 for Library Management System

Q.5 Attempt any TWO of the following : [16]

Q.5(a) What is software? What are its characteristics? [8]

Ans.: Software is

- (1) Instructions (computer programs) that when executed provide desired function and performance,
- (2) Data structures that enable the programs to adequately manipulate information, and
- (3) Documents that describe the operation and use of the programs

Software is written to handle an Input - Process - Output system to achieve predetermined goals. Software is logical rather than a physical system element.

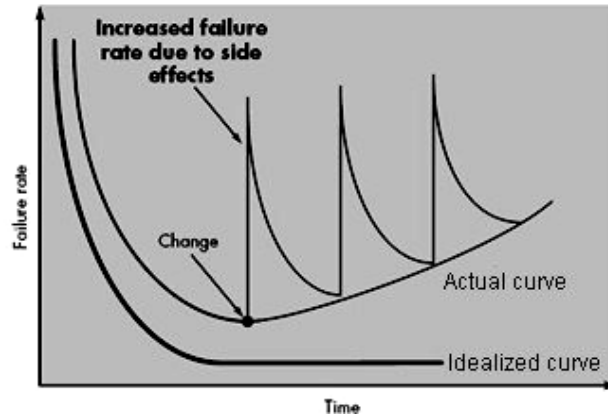
SOFTWARE CHARACTERISTICS :

a) **Software is developed or engineered; it is not manufactured in the classical sense.**

- Although some similarities exist between software development and hardware manufacture, the two activities are fundamentally different.
- In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are nonexistent (or easily corrected) for software.
- Both activities are dependent on people, but the relationship between people applied and work accomplished is entirely different.
- Software costs are concentrated in engineering. This means that software projects cannot be managed as if they were manufacturing projects.

b) **Software costs are concentrated in engineering. This means that software projects cannot be managed as if they were manufacturing projects.**

- Software is not susceptible to the environmental maladies that cause hardware to wear out.



- In theory, therefore, the failure rate curve for software should take the form of the "idealized curve" shown in the figure. Undiscovered defects will cause high failure rates early in the life of a program. However, these are corrected (ideally, without introducing other errors) and the curve flattens as shown.
 - The idealized curve is a gross oversimplification of actual failure models for software. However, the implication is clear—software doesn't wear out. But it does deteriorate! This seeming contradiction can best be explained by considering the "actual curve" shown in Figure. During its life, software will undergo change (maintenance). As changes are made, it is likely that some new defects will be introduced, causing the failure rate curve to spike as shown in Figure.
 - Before the curve can return to the original steady-state failure rate, another change is requested, causing the curve to spike again. Slowly, the minimum failure rate level begins to rise—the software is deteriorating due to change.
- c) **Although the industry is moving toward component- based assembly, most software continues to be custom built.**
- The reusable components have been created so that the engineer can concentrate on the truly innovative elements of a design, that is, the parts of the design that represent something new.
 - In the software world, it is something that has only begun to be achieved on a broad scale. A software component should be designed and implemented so that it can be reused in many different programs.

Q.5(b) What are major task of requirement engineering?

[8]

Ans.: Requirements engineering tasks

1. **Inception:** Inception means beginning. It is usually said that requirement engineering is a—communication intensive activity. The customer and developer meet and they the overall scope and nature of the problem statements. By having proper inception phase the developer will have clear idea about the system and as a result of that better understanding of a system can be achieved. Once the system is clear to the developer they can implement a system with better efficiency.
2. **Elicitation:** Elicitation task will help the customer to define the actual requirement of a system. To know the objectives of the system or the project to be developed is a critical job. This phase will help people to determine the goal of a system and clear idea about the system can be achieved.
3. **Elaboration:** The information obtained from the customer during inception and elicitation is expanded and refined during elaboration. This requirement engineering activity focuses on developing a refined technical model of software functions, features and constraints.
4. **Negotiation:** This phase will involve the negotiation between what user actual expects from the system and what is actual feasible for the developer to build. Often it is seen

that user always expect lot of things from the system for lesser cost. But based on the other aspect and feasibility of a system the customer and developer can negotiate on the few key aspect of the system and then they can proceed towards the implementation of a system.

5. **Specification:** A specification can be a re-written document, a set of graphical models, a formal mathematical model, a collection of usage scenario, a prototype, or any combinations of these.

The specification is the final work product produced by the requirement engineers. It serves as the foundation for subsequent software engineering activities. It describes the function and performance of a computer based system and the constraints that will govern its development.

6. **Validation:** The work products produced as a consequence of requirements engineering are assessed for quality during a validation step. Requirements validation examines the specification to ensure that all software requirements have been stated unambiguously; that inconsistencies, omissions and errors have been detected and corrected, and that the work products conform to the standards established for the process, the project, and the product.
7. **Requirements management :** Requirement management begins with identification. Each requirement is assigned a unique identifier. Once requirement have been identified, traceability tables are developed.

Q.5(c) Explain the term debugging. Explain different debugging.

[8]

Ans.: Debugging occurs as a consequence of successful testing. That is, when a test case uncovers an error, debugging is the process that results in the removal of the error.

Although debugging can and should be an orderly process, it is still very much an art.

There are three Different Debugging Strategies available as follows :

- (1) Brute Force,
- (2) Backtracking, and
- (3) Cause Elimination.

1. **Brute Force:** This category of debugging is probably the most common and least efficient method for isolating the cause of a software error. Brute force debugging methods are applied when all else fails. Using a "let the computer find the error" philosophy, memory dumps are taken, run-time traces are invoked, and the program is loaded with WRITE statements. In the morass of information that is produced a clue is found that can lead us to the cause of an error. Although the mass of information produced may ultimately lead to success, it more frequently leads to wasted effort and time. Thought must be expended first.
2. **Backtracking:** It is a fairly common debugging strategy that can be used successfully in small programs. Beginning at the site where a symptom has been uncovered, the source code is traced backward (manually) until the site of the cause is found. Unfortunately, as the number of source lines increases, the number of potential backward paths may become unmanageably large.
3. **Cause Elimination:** It is manifested by induction or deduction and introduces the concept of binary partitioning. Data related to the error occurrence are organized to isolate potential causes. A "cause hypothesis" is devised and the aforementioned data are used to prove or disprove the hypothesis. Alternatively, a list of all possible causes is developed and tests are conducted to eliminate each. If initial tests indicate that a particular cause hypothesis shows promise, data are refined in an attempt to isolate the bug.

Q.6 Attempt any FOUR of the following : [16]

Q.6(a) Explain Deployment principles. [4]

Ans.: Deployment principle:

Principle 1: Manage customer's expectations.

It always happens that customer wants more than he has started earlier as his requirements. It may be the case that customer gets disappointed, even after getting all his requirements satisfied. Hence at time of delivery developer must have skills to manage customer's expectations.

Principle 2: Assembly and test complete delivery package.

It is not the case that the deliverable package is 'only software'. The customer must get all supporting and essential help from developer's side.

Principle 3: Record-keeping mechanism must be established for customer support.

Customer support is important factor in deployment phase. If proper support is not provided, customer will not be satisfied. Hence support should be well planned and with record-keeping mechanism.

Principle 4: Provide essential instructions, documentations and manual.

Many times, developer thinks —when project is successful deliverable part is only working program. But reality is that working program is just part of software product. Actual project delivery includes all documentations, help files and guidance for handling the software by user.

Principle 5: Don't deliver any defective or buggy software to the customer.

In incremental type of software, software organizations may deliver some defective software to the customer by giving assurance that the defects will be removed in next increment.

Q.6(b) Differentiate between validation and verification. [4]

Ans.:

	Validation	Verification
1.	Validation is a dynamic mechanism of validating and testing	Verification is a static practice of verifying documents, design, code and
2.	It always involves executing the code.	It does not involve executing the code.
3.	It is computer based execution of program.	It is human based checking of documents and files.
4.	Validation uses methods like black box (functional) testing, gray box testing, and white box (structural)	Verification uses methods like inspections, reviews, walkthroughs, and Desk-checking etc.
5.	Validation is to check whether software meets the customer	Verification is to check whether the software conforms to specifications.
6.	It can catch errors that verification cannot catch. It is High Level	It can catch errors that validation cannot catch It is low level exercise.
7.	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
8.	Validation is earned 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 follow's after verification.	It generally comes before Validation

Q.6(c) Explain about software quality assurance. [4]

Ans.: Software quality assurance (SQA) is a process that ensures that developed software meets and complies with defined or standardized quality specifications. SQA is an ongoing process within the software development life cycle (SDLC) that routinely checks the developed software to ensure it meets desired quality measures. SQA helps ensure the development of high-quality software. SQA practices are implemented in most types of software development, regardless of the underlying software development model being used. In a broader sense, SQA incorporates and implements software testing methodologies to test software. Rather than checking for quality after completion, SQA processes test for quality in each phase of development until the software is complete.

With SQA, the software development process moves into the next phase only once the current/previous phase complies with the required quality standards.

SQA generally works on one or more industry standards that help in building software quality guidelines and implementation strategies. These standards include the ISO 9000 and capability maturity model integration (CMMI).

Software quality assurance is composed of a variety of tasks associated with two different constituencies - the software engineers who do technical work and an SQA group that has responsibility for quality assurance planning, oversight, record keeping, analysis, and reporting. Software engineers address quality (and perform quality assurance and quality control activities) by applying solid technical methods and measures, conducting formal technical reviews, and performing well-planned software testing.

Q.6(d) Describe behavioral model. [4]

Ans.: Behavioral models are used to describe the overall behavior of a system.

The behavioral model indicates how software will respond to external events or stimuli.

To create the model, you should perform the following steps:

1. Evaluate all use cases to fully understand the sequence of interaction within the system.
2. Identify events that drive the interaction sequence and understand how these events relate to specific objects.
3. Create a sequence for each use case.
4. Build a state diagram for the system.
5. Review the behavioral model to verify accuracy and consistency

Two types of behavioral model are:

Data processing models that show how data is processed as it moves through the system

State machine models that show the systems response to events

These models show different perspectives, so both of them are required to describe the system's behavior

Data Processing Model

Data flow diagrams (DFDs) may be used to model the system's data processing. These show the processing steps as data flows through a system. DFDs are an intrinsic part of many analysis methods. It is Simple and intuitive notation that customers can understand. It show end-to-end processing of data. DFDs model the system from a functional perspective. It is helpful to develop an overall understanding of the system.

State machine Model

These model the behavior of the system in response to external and internal events. They show the system's responses to stimuli so are often used for modeling real modeling real time systems. State machine models show system states as nodes and events as arcs between these nodes. When an event occurs, the system moves from one state to another. State charts, An integral part of the UML are used to represent state machine models

Q.6(e) State testing principles and coding principles.

[4]

Ans.: **Testing Principles :** The objective of testing is to find undiscovered error.

- (i) All tests should be traceable to customer requirements.
- (ii) **Tests should be planned long before testing begins :** Testing planning can start as soon as analysis is started. Detailed test cases can begin as soon as design model is generated.
- (iii) **The Pareto principle applies to software testing :** The principle states that 80% of all errors uncovered during testing will likely be traceable to 20% of all program components. These components can be can be separated and tested thoroughly.
- (iv) **Testing should begin "in the small" and progress toward testing "in the large" :** Start with unit testing and end with system testing.
- (v) Exhaustive testing is not possible.

Coding Principles and Concepts

The construction activity represents coding and testing of the application. Today code can be written or generated using tools.

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

- Understand the problem you're trying to solve.
- Understand basic design principles and concepts.
- Pick a right programming language.
- Select a programming environment that provides tools that will make your work easier.
- Create a set of unit tests that can be used to test the component when it is ready.

Coding Principles : As you begin writing code, be sure you

- Constrain your algorithms by following structured programming practice.
- Select data structures that will meet the design requirements.
- 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.
- Use indentation to make programs readable.

Validation principles : After completing first coding pass, be sure you

- Conduct code walkthrough.
- Perform unit test and correct errors.

Testing Principles : The objective of testing is to find undiscovered error.

- (i) All tests should be traceable to customer requirements.
- (ii) **Tests should be planned long before testing begins :** Testing planning can start as soon as analysis is started. Detailed test cases can begin as soon as design model is generated.
- (iii) **The Pareto principle applies to software testing :** The principle states that 80% of all errors uncovered during testing will likely be traceable to 20% of all program components. These components can be can be separated and tested thoroughly.
- (iv) **Testing should begin "in the small" and progress toward testing "in the large" :** Start with unit testing and end with system testing.
- (v) Exhaustive testing is not possible.

Q.6(f) Explain SCM.

[4]

Ans.: 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.

SCM is an umbrella activity that is applied throughout the software process. SCM is a set of tracking and control activities that are initiated when SE project begin and terminate only when the software is taken out of operation. SCM helps to improve software quality and on time delivery. SCM defines the project strategy for change management. When formal SCM is invoked, the change control process produces software change requests, reports and engineering change orders. SCM helps to track, analyze and control every work product.

Need of SCM

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

