Important Instructions to examiners:

1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling errors should not be given more Importance (Not applicable for subject English and Communication Skills).
4) While assessing figures, examiner may give credit for principal components indicated in the figure. The figures drawn by candidate and model answer may vary. The examiner may give credit for any equivalent figure drawn.
5) Credits may be given step wise for numerical problems. In some cases, the assumed constant values may vary and there may be some difference in the candidate's answers and model answer.
6) In case of some questions credit may be given by judgment on part of examiner of relevant answer based on candidate's understanding.
7) For programming language papers, credit may be given to any other program based on equivalent concept.

1. A) Attempt any three. **Marks 12**

a) Write a note on current trends of software development.

(Each Point-1 Mark (Any four))

(Any relevant current trends of software's must be considered)

Ans:

- What Factors Determine the Success of a Trend? What characterizes successful technological trends: Their technical merit? Their ability to open new markets? Their ability to alter the economics of existing markets?
- What Lifecycle Does a Trend Follow? Whereas the traditional view is that trends evolve along a well defined, predictable lifecycle that proceeds from a research idea to a finished product through a transfer process, we find that many current trends have either short circuited this cycle or followed another one.
- How Early Can a Successful Trend Be Identified? If we know how to identify success factors, and/or we understand the lifecycle of a trend, then we seek to identify early signs of success of a trend. Theoretically, we seek the ability to recognize the next trend ahead of everybody else.
- What Aspects of Evolution Are Controllable? Can corporations use their market clout to impose trends? Can the government use its resources to impose trends? What role do standards play in defining trends? The careful analysis of Ada vs. Java, for example, should be enlightening in this regard.

b) Explain testing strategy for object oriented architecture.
(Each Point-1 Mark (Any four))

Ans:

- With object-oriented software, you can no longer test a single operation in isolation (conventional thinking)
- Traditional top-down or bottom-up integration testing has little meaning
- Class testing for object-oriented software is the equivalent of unit testing for conventional software
  - Focuses on operations encapsulated by the class and the state behavior of the class
- Drivers can be used
  - To test operations at the lowest level and for testing whole groups of classes
  - To replace the user interface so that tests of system functionality can be conducted prior to implementation of the actual interface
- Stubs can be used
  - In situations in which collaboration between classes is required but one or more of the collaborating classes has not yet been fully implemented
- Two different object-oriented testing strategies
  - Thread-based testing
- Integrates the set of classes required to respond to one input or event for the system
- Each thread is integrated and tested individually
- Regression testing is applied to ensure that no side effects occur
  - Use-based testing
- First tests the independent classes that use very few, if any, server classes
- Then the next layer of classes, called dependent classes, are integrated
- This sequence of testing layer of dependent classes continues until the entire system is constructed.

c) State four causes of failed software projects.
((Each Point-1 Mark (Any four))

Ans: Reasons for Failure Citations:

1) Planning - Insufficient planning
2) Scope- Poorly defined requirements and scope
3) Stakeholder -Ineffective stakeholder management
4) Commitment - Lack of executive support or project sponsorship
5) Time - Poor estimation and/or scheduling; unrealistic deadlines
6) Cost - Unrealistic budget
7) Quality and Testing- Shortened or elimination of quality assurance practices; inadequate testing or reduction of testing time within the software development lifecycle;
8) Resources – Insufficient qualified resources
9) Communication- Poor or ineffective communication
10) Risk - Insufficient risk planning and risk management
11) Politics - Stakeholder politics; user politics; corporate
12) Stakeholder - Lack of user and stakeholder involvement
13) Project manager- Poor project management or project manager
14) Process - Undefined development processes or lack of adherence to process standards (Sloppy development practices); failure to implement best practices and lessons learned
15) Product - Speed to market, desire for innovation; deliver now and correct errors later
16) Project Goals - Unrealistic or unarticulated project goals
17) Change Control - Poorly managed change control
18) Technology – changes in technology

d) Briefly explain ISO 9001: 2000 software quality standards.
((Each Point-1 Mark (Any four))

Ans: The ISO 9000 quality assurance models treat an enterprise as a network of interconnected processes.

- For a quality system to be ISO compliant, these processes must address the areas identified in the standard and must be documented and practiced as described.
- ISO 9000 describes the elements of a quality assurance system in general terms.

- These elements include the organizational structure, procedures, processes, and resources needed to implement quality planning, quality control, quality assurance, and quality improvement.
- However, ISO 9000 does not describe how an organization should implement these quality system elements. Consequently, the challenge lies in designing and implementing a quality assurance system that meets the standard and fits the company's products, services, and culture.
- The ISO 9001 Standard ISO 9001 is the quality assurance standard that applies to software engineering.
- The standard contains 20 requirements that must be present for an effective quality assurance system.
- Because the ISO 9001 standard is applicable to all engineering disciplines, a special set of ISO guidelines (ISO 9000-3) have been developed to help interpret the standard for use in the software process.
- The requirements delineated by ISO 9001 address topics such as management responsibility, quality system, contract review, design control, document and data control, product identification and traceability, process control, inspection and testing, corrective and preventive action, control of quality records, internal quality audits, training, servicing, and statistical techniques.
- In order for a software organization to become registered to ISO 9001, it must establish policies and procedures to address each of the requirements just noted (and others) and then be able to demonstrate that these policies and procedures are being followed.

B) Attempt any one. **Marks 06**

a) State and describe six communication principles.
(Only State- 2 Marks, Or Each Point with explanation -1 Mark (Any six))

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 jargon. If you have responsibility for conducting a meeting, prepare an agenda in advance of the meeting.

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

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

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

Principle 6. Strive for collaboration. Collaboration and consensus occur when the collective knowledge of members of the team is used to describe product or system functions or features. Each small 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 conversation modular; leaving one topic only after it has been resolved.

Principle 8. If something is unclear, draw a picture. Verbal communication goes only so far. A sketch or drawing can often provide clarity when words fail to do the job.

Principle 9. (a) Once you agree to something, move on. (b) If you can't agree to something, move on. (c) If a feature or function is unclear and cannot be clarified at the moment, move on.

Communication, like any software engineering activity, takes time. Rather than iterating endlessly,

The people who participate should recognize that many topics require discussion (see Principle 2) 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.

b) Describe seven distinct functions of requirement engineering.
(Each Point-1 Mark)

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 decide 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, and a collection of usage scenario, a prototype, or any combinations of these. The specification is the final work product produced by the requirement engineers. It 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

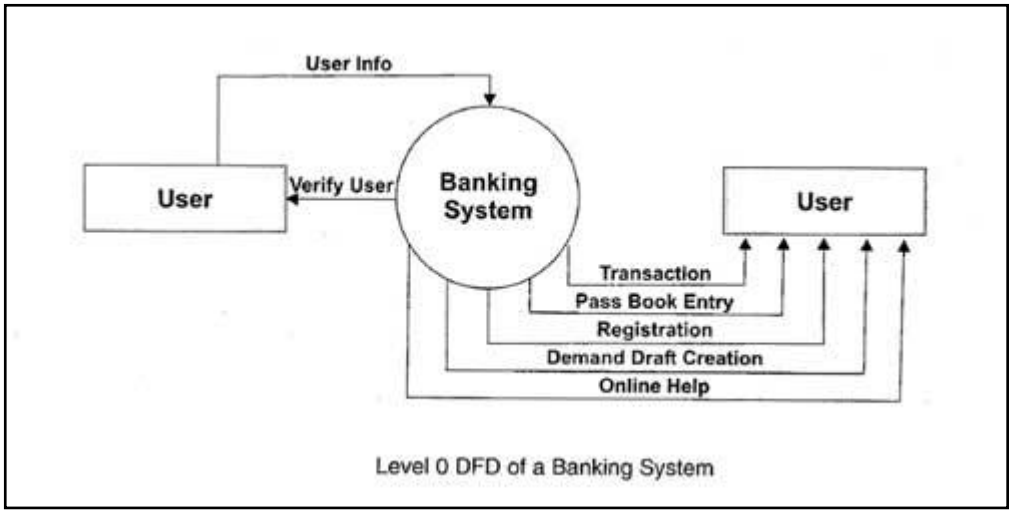
Requirement management begins with identification. Each requirement is assigned a unique identifier. Once requirement have been identified, traceability tables are developed.

2. Attempt any four. **Marks 16**

a) Describe a simple DFD for cash withdrawal by an account holder from saving Bank.

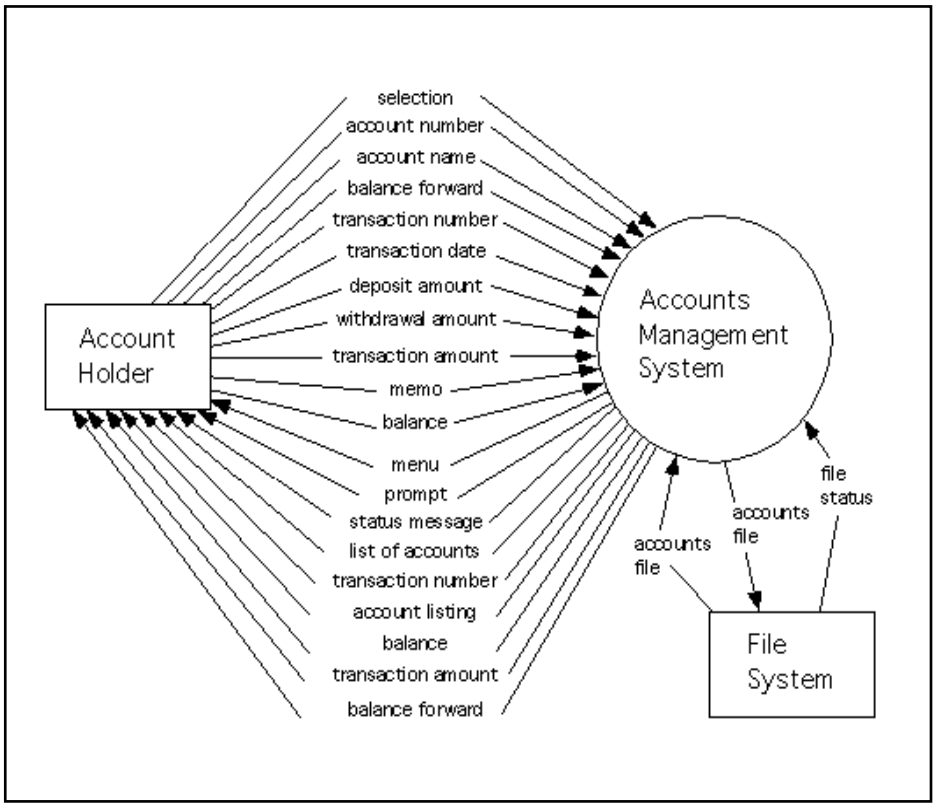
(Description- 1 Mark, Any relevant Diagram – 3 Marks)

Ans: DFD for cash withdrawal by an account holder from saving Bank.



Level 0 DFD of a Banking System

OR



b) State and describe any four characteristics of a testable software.
((Only State- 2 Marks, Or Each Point with explanation -1 Mark (Any Four))

Ans: Testability:

Software testability is simply how easily [a computer program] can be tested.

Operability

– "The better it works, the more efficiently it can be tested."

– The system has few bugs (bugs add analysis and reporting overhead to the test process).

– No bugs block the execution of tests.

– The product evolves in functional stages (allows simultaneous development and testing)

Observability

– "What you see is what you test."

– Distinct output is generated for each input.

– System states and variables are visible or queriable during execution.

– Past system states and variables are visible or queriable (e.g., transaction logs).

– All factors affecting the output are visible.

– Incorrect output is easily identified.

– Internal errors are automatically detected through self-testing mechanisms.

– Internal errors are automatically reported.

– Source code is accessible

Simplicity

– "The less there is to test, the more quickly we can test it."

– Functional simplicity (e.g., the feature set is the minimum necessary to meet requirements)

– Structural simplicity (e.g., architecture is modularized to limit the propagation of faults).

– Code simplicity (e.g., a coding standard is adopted for ease of inspection and maintenance)

Stability

– "The fewer the changes, the fewer the disruptions to testing."

– Changes to the software are infrequent.

– Changes to the software are controlled.

– Changes to the software do not invalidate existing tests.

– The software recovers well from failures.

Decomposability

– "By controlling the scope of testing, we can more quickly isolate the errors and conduct smart testing.

– The software system is built from independent modules.

– Software modules can be tested independently

Understandability

– "The more information we have, the smarter we will test."

– The design is well understood.

– Dependencies between internal, external, and shared components are well understood.

– Changes to the design are communicated.

– Technical documentation is instantly accessible.

– Technical documentation is well organized.

– Technical documentation is specific and detailed.

– Technical documentation is accurate

Controllability

– "The better we can control the software, the more the testing can be optimized—
– All possible outputs can be generated through some combination of input.
– All code is executable through some combination of input.
– Software and hardware states and variables can be controlled directly by the test engineer.
– Input and output formats are consistent and structured.
– Tests can be conveniently specified, automated, and reproduced.

c) State decomposition technique and list its types. Explain one type in detail.
(Definition – 1 Mark, Type -1 mark, Description -2 Marks)

Ans: Three different points of view for the decomposition approach Software Sizing, Decomposition of the problem, and Decomposition of the process.

1. Software Sizing

- Size refers to a quantifiable outcome of the software project.
- If a direct approach is taken, size can be measured in LOC.
- If an indirect approach is chosen, size is represented as FP.
- Four different approaches to the sizing problem:

a. "Fuzzy logic" sizing:

This approach uses the approximate reasoning techniques that are the cornerstone of fuzzy logic.

To apply this approach, the planner must identify the type of application, establish its magnitude on a qualitative scale, and then refine the magnitude within the original range.

b. Function point sizing:

The planner develops estimates of the information domain characteristics.

c. Standard component sizing:

Software is composed of a number of different ―standard components‖ that are generic to a particular application area.

For example, the standard components for an information system are screens, reports, batch programs, files.

The project planner estimates the number of occurrences of each standard component and then uses historical project data to determine the delivered size per standard component.

d. Change sizing:

This approach is used when a project encompasses the use of existing software that must be modified in some way as part of a project.

The planner estimates the number and type (e.g., reuse, adding code, changing code, and deleting code) of modifications that must be accomplished.

Using an ―effort ratio for each type of change, the size of the change may be estimated.

2. Problem decomposition

– Example of baseline productivity metrics are LOC/pm or FP/pm

– Making the use of single baseline productivity metric is discouraged

– The planner may choose another component for sizing such as classes or objects.

– In general, LOC/pm or FP/pm averages should be computed by project domain

– Local domain averages should be used

– Statistical approach – three-point or expected-value estimate

$S = (s_{opt} + 4s_m + s_{pess})/6$

S = expected-value for the estimation variable (size)

s_{opt}= optimistic value

s_m= most likely value

s_{pess}= pessimistic value

Example LOC-Based Estimation and FP-Based Estimation

3. Process decomposition

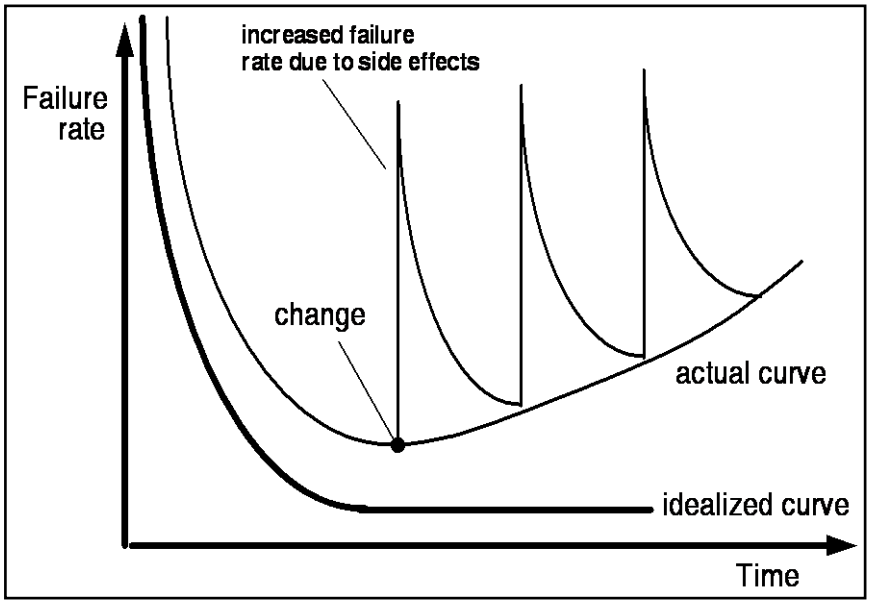
– Based on the types of process used in the project and size and complexity of the project estimate the best process model suitable for the project.

– If the project deadline is tight go for incremental model.

– If the project deadline is very tight go for RAD model.

d) Describe the failure curve for software and list any four software characteristics
(Failure curve – 1 Mark, Description -1 Mark, list- 2 Marks)

Ans:



The failure rate curve for software should take the form of the "idealized curve" shown in 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.

Following are the characteristics of Software

1. Software is developed or engineered; it is not manufactured in the classical sense.
2. Software doesn't "wear out".
3. Although the industry is moving toward component-based assembly, most software continues to be custom built.

e) What is the meaning of planning? State four planning principles.
(Meaning – 1 Mark, Any four planning principles – 3 Marks)

Ans: The communication activity helps you to define your overall goals and objectives (subject, of course, to change as time passes). However, understanding these goals and objectives is not the same as defining a plan for getting there. The planning activity encompasses a set of management and technical practices that enable the software team to define a road map as it travels toward its strategic goal and tactical objectives.

Principle 1. Understand the scope of the project. It's impossible to use a road map if you don't know where you're going. Scope provides the software team with a destination.

Principle 2. Involve stakeholders in the planning activity. Stakeholders define priorities and establish project constraints. To accommodate these realities, software engineers must often negotiate order of delivery, time lines, and other project-related issues.

Principle 3. Recognize that planning is iterative. A project plan is never engraved in stone. As work begins, it is very likely that things will change. As a consequence, the plan must be adjusted to accommodate these changes. In addition, iterative, incremental process models dictate replanning after the delivery of each software increment based on feedback received from users.

Principle 4. Estimate based on what you know. The intent of estimation is to provide an indication of effort, cost, and task duration, based on the team's current understanding of the work to be done. If information is vague or unreliable, estimates will be equally unreliable.

Principle 5.Consider risk as you defines the plan. If you have identified risks that have high impact and high probability, contingency planning is necessary. In addition, the project plan (including the schedule) should be adjusted to accommodate the likelihood that one or more of these risks will occur.

Principle 6. Be realistic. People don't work 100 percent of every day. Noise always enters into any human communication. Omissions and ambiguity are facts of life. Change will occur. Even the best software engineers make mistakes. These and other realities should be considered as a project plan is established.

Principle 7.Adjust granularity as you defines the plan. Granularity refers to the level of detail that is introduced as a project plan is developed. A "high-granularity" plan provides significant work task detail that is planned over relatively short time increments (so that tracking and control occur frequently). A "low-granularity" plan provides broader work tasks that are planned over

longer time periods. In general, granularity moves from high to low as the project time line moves away from the current date. Over the next few weeks or months, the project can be planned in significant detail. Activities that won't occur for many months do not require high granularity (too much can change).

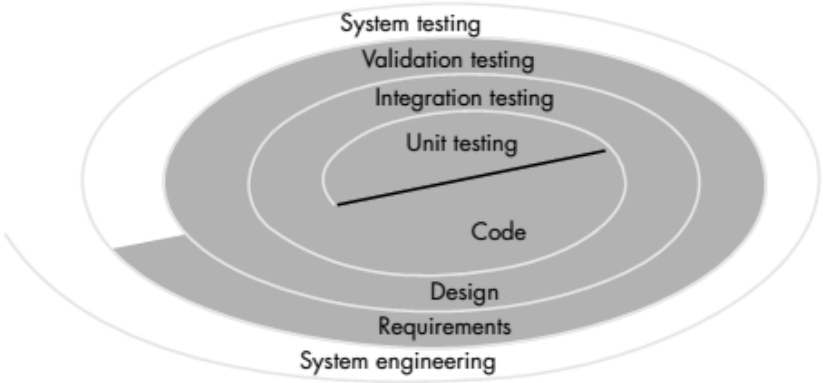
Principle 8. Define how you intend to ensure quality. The plan should identify how the software team intends to ensure quality. If technical reviews are to be conducted, they should be scheduled. If pair programming is to be used during construction, it should be explicitly defined within the plan.

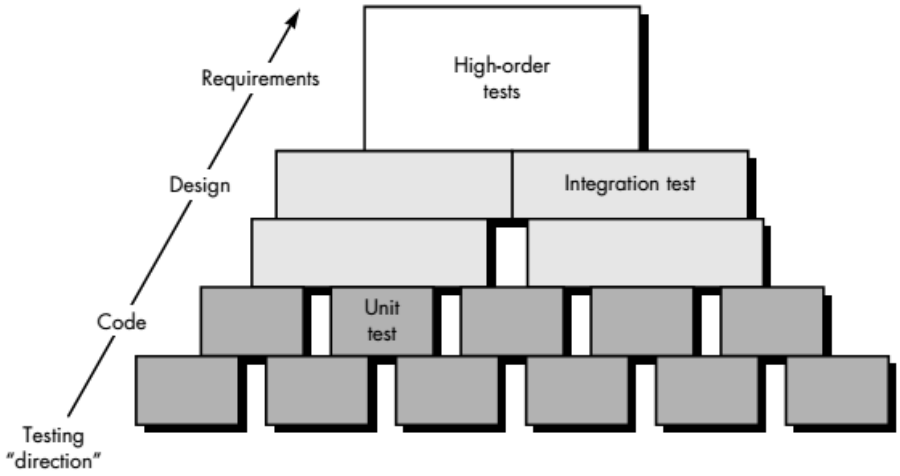
Principle 9. Describe how you intend to accommodate change. Even the best planning can be obviated by uncontrolled change. You should identify how changes are to be accommodated as software engineering work proceeds. For example, can the customer request a change at any time? If a change is requested, is the team obliged to implement it immediately? How is the impact and cost of the change assessed?

Principle 10.Track the plan frequently and make adjustments as required. Software projects fall behind schedule one day at a time. Therefore, it makes sense to track progress on a daily basis, looking for problem areas and situations in which scheduled work does not conform to actual work conducted. When slippage is encountered, the plan is adjusted accordingly.

f) Describe in brief four level testing process in test execution.
(Diagram -1 Mark, Description - 3 Marks)

Ans:





Initially, system engineering defines the role of software and leads to software requirements analysis, where the information domain, function, behavior, performance, constraints, and validation criteria for software are established. Moving inward along the spiral, we come to design and finally to coding. To develop computer software, we spiral inward along streamlines that decrease the level of abstraction on each turn.

A strategy for software testing may also be viewed in the context of the spiral (Figure). Unit testing begins at the vortex of the spiral and concentrates on each unit (i.e., component) of the software as implemented in source code. Testing progresses by moving outward along the spiral to integration testing, where the focus is on design and the construction of the software architecture. Taking another turn outward on the spiral, we encounter validation testing, where requirements established as part of software requirements analysis are validated against the software that has been constructed. Finally, we arrive at system testing, where the software and other system elements are tested as a whole. To test computer software, we spiral out along streamlines that broaden the scope of testing with each turn.

Considering the process from a procedural point of view, testing within the context of software engineering is actually a series of four steps that are implemented sequentially. The steps are shown in Figure. Initially, tests focus on each component individually, ensuring that it functions properly as a unit. Hence, the name unit testing. Unit testing makes heavy use of white-box testing techniques, exercising specific paths in a module's control structure to ensure complete coverage and maximum error detection. Next, components must be assembled or integrated to form the complete software package. Integration testing addresses the issues associated with the

dual problems of verification and program construction. Black-box test case design techniques are the most prevalent during integration, although a limited amount of white-box testing may be used to ensure coverage of major control paths. After the software has been integrated (constructed), a set of high-order tests are conducted. Validation criteria (established during requirements analysis) must be tested. Validation testing provides final assurance that software meets all functional, behavioral, and performance requirements. Black-box testing techniques are used exclusively during validation.

The last high-order testing step falls outside the boundary of software engineering and into the broader context of computer system engineering. Software, once validated, must be combined with other system elements (e.g., hardware, people, databases). System testing verifies that all elements mesh properly and that overall system function/performance is achieved.

3. Attempt any for. Marks 16

a) State any four sources of Change in change Management.
(Each source -1 Mark)

Ans: The fundamental sources of change are:

- New business or market conditions dictate changes in product requirements or business needs.
- New customer needs demand modification of data produced by information systems, functionality delivered by products or delivered by computer based systems.
- Reorganization or business growth/ downsizing cause's changes in product priorities or software engineering team structure.
- Budgetary or scheduling constraints cause a redefinition of the system or product.

b) Describe smoke testing.
(Each Point -1 Mark (Any Four)

Ans:

- Smoke testing is an integration testing approach that is commonly used when "shrink wrapped" software products are being developed. It is designed as a pacing mechanism for time-critical projects, allowing the software team to assess its project on a frequent basis. In essence, the smoke testing approach encompasses the following activities:

- Software components that have been translated into code are integrated into a "build." A build includes all data files, libraries, reusable modules, and engineered components that are required to implement one or more product functions.
- A series of tests is designed to expose errors that will keep the build from properly performing its function. The intent should be to uncover "show stopper" errors that have the highest likelihood of throwing the software project behind schedule.
- The build is integrated with other builds and the entire product (in its current form) is smoke tested daily. The integration approach may be top down or bottom up.
- The daily frequency of testing the entire product may surprise some readers. However, frequent tests give both managers and practitioners a realistic assessment of integration testing progress. McConnell describes the smoke test in the following manner:
- The smoke test should exercise the entire system from end to end. It does not have to be exhaustive, but it should be capable of exposing major problems. The smoke test should be thorough enough that if the build passes, you can assume that it is stable enough to be tested more thoroughly.
- Smoke testing provides a number of benefits when it is applied on complex, time critical software engineering projects:
- Integration risk is minimized. Because smoke tests are conducted daily, incompatibilities and other show-stopper errors are uncovered early, thereby reducing the likelihood of serious schedule impact when errors are uncovered. The quality of the end-product is improved. Because the approach is construction (integration) oriented, smoke testing is likely to uncover both functional errors and architectural and component-level design defects. If these defects are corrected early, better product quality will result.
- Error diagnosis and correction are simplified. Like all integration testing approaches, errors uncovered during smoke testing are likely to be associated with "new software increments"—that is, the software that has just been added to the build(s) is a probable cause of a newly discovered error.
- Progress is easier to assess. With each passing day, more of the software has been integrated and more has been demonstrated to work. This improves team morale and gives managers a good indication that progress is being made.

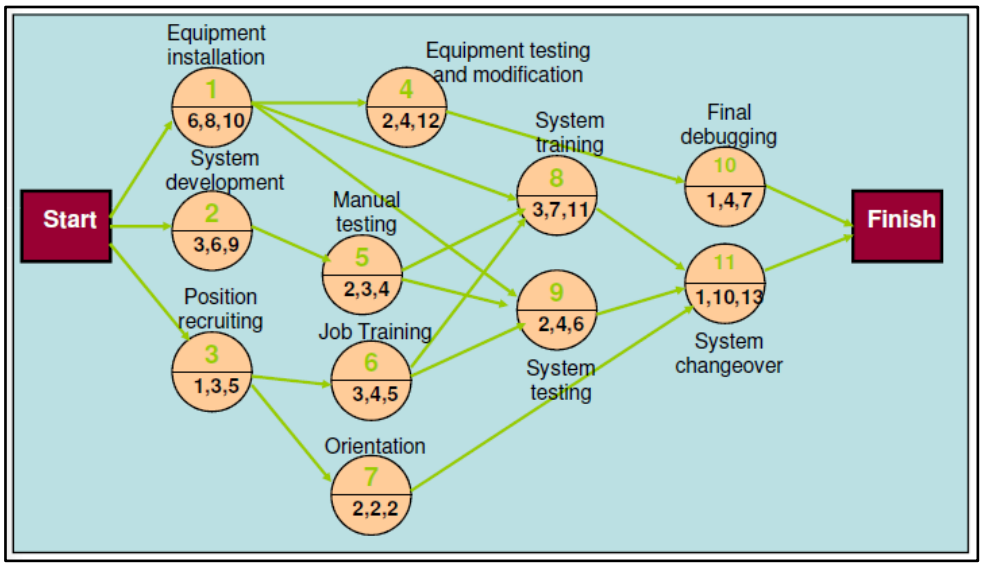
c) Describe relationship between effort and delivery time using a graph in project scheduling.

(Relationship between effort and delivery- 2 Marks; Any relevant example- 2 Marks)

Ans: The relationship between effort and delivery time using graph in project scheduling gives amount of efforts to be put-in to complete the task so that the deadline can be met.

Usually the project manager draws a graph where each node defines distinct activity and then these activities given some duration depending on the deadline of a project. The project manager keeps on verifying the progress of the project.

Consider following example:



As shown in above diagram there are various nodes / tasks assigned to various teams; each of the node has its own duration and the dependency amongst two task /nodes. Whenever any task lags behind the prescribed schedule then extra efforts can be given to ensure that the work can be completed in given period of time.

d) Enlist four recommended guidelines to avoid schedule failure.

(Any four guidelines are expected -1 Mark each)

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

1. An unrealistic deadline established by someone outside the software development group and forced on managers and practitioners within the group.

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

e) Describe SQA and activities involved.

(Description of SQA - 1 Mark; Activities of SQA- 3 Marks)

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

Prepare an SQA plan for a project. The plan is developed during project planning and is reviewed by all interested parties. Quality assurance activities performed by the software engineering team and the SQA group are governed by the plan. The plan identifies

- evaluations to be performed
- audits and reviews to be performed
- standards that are applicable to the project
- procedures for error reporting and tracking
- documents to be produced by the SQA group
- amount of feedback provided to the software project team

Participate in the development of the project's software process description.

The software team selects a process for the work to be performed. The SQA group reviews the process description for compliance with organizational policy, internal software standards, externally imposed standards (e.g., ISO-9001), and other parts of the software project plan.

Review software engineering activities to verify compliance with the defined software process. The SQA group identifies, documents, and tracks deviations from the process and verifies that corrections have been made.

Audits designated software work products to verify compliance with those defined as part of the software process. The SQA group reviews selected work products; identifies, documents, and tracks deviations; verifies that corrections have been made; and periodically reports the results of its work to the project manager.

Ensure that deviations in software work and work products are documented and handled according to a documented procedure. Deviations may be encountered in the project plan, process description, applicable standards, or technical work products.

Records any noncompliance and reports to senior management. Noncompliance items are tracked until they are resolved.

4. A) Attempt any three. Marks 12

a) Describe software estimation.
(Any four point - 1 Mark each)

Ans:

- Software Project Estimation basically includes estimation of cost and effort estimation.
- Variables like human, technical, environmental, and political—can affect the ultimate cost of software and effort applied to develop it.
- Software project estimation can be transformed from a black art to a series of systematic steps that provide estimates with acceptable risk.
- To achieve reliable cost and effort estimates, a number of options arise:
- Delay estimation until late in the project (obviously, we can achieve 100% accurate estimates after the project is complete!).
- Base estimates on similar projects that have already been completed.
- Use relatively simple decomposition techniques to generate project cost and effort estimates.
- Use one or more empirical models for software cost and effort estimation.
- Decomposition techniques take a "divide and conquer" approach to software project estimation. By decomposing a project into major functions and related software engineering activities, cost and effort estimation can be performed in a stepwise fashion.

- Empirical estimation models can be used to complement decomposition techniques and offer a potentially valuable estimation approach in their own right. A model is based on experience (historical data) and takes the form.

b) What is meant by automated debugging? How it assists in isolating software errors?
(Automated Debugging- 2 Marks; Isolating software errors- 2 Marks)

Ans: An automated debugging is a process of debugging the system / module automatically. In such debugging methods the software tester writes a debugging scripts/code and this code can be use repetitively to test similar module.

The script can be executed number of times to get the better accuracy and precision. The best part of automated debugging is that it is done by computer itself with the logic being provided by the tester. It is time saving and ensures that more number of modules can be tested in given time.

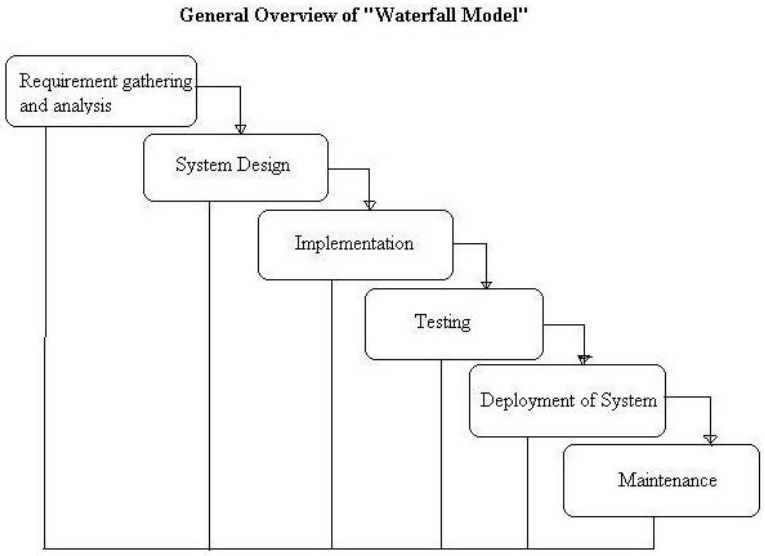
Isolating Software Error:

Whenever the programming code does not fulfill the condition provided by the tester's script it generates and error and displays it to the tester. Then tester takes appropriate measurement to overcome such error.

c) Describe the waterfall model of process development.

(Diagram -1 Mark; Description -3 Marks; Advantage/disadvantage optional)

Ans: The Waterfall Model was first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed fully before the next phase can begin. At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project. In waterfall model phases do not overlap.



Phases in Waterfall Model:-

Requirement Gathering and analysis:-

In this phase the analysis team analyzes the problem statement. The do market survey and tries to gather more and more information for the system.

System Design:-

In this phase he Design team draws various diagram for the system and gives the graphical representation of the system.

Implementation:-

In this phase system is actually gets implemented i.e. coding of the system is done.

Testing: -

In this phase the testing team tries to test the system and finds the bugs (if any)

Deployment of System:-

In this phase the system is actually gets deployed on the client side.

Maintenance:-

This phase ensures that the proper maintenance of the system is provided and regular updates are provided.

Advantages of waterfall model:

- Simple and easy to understand and use.
- Easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.

Disadvantages of waterfall model:

- Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.

d) Differentiate any four points differentiate between Black box and White box testing.

(Each difference-1 Mark for)

Ans:

Criteria	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

B) Attempt any one. **Marks 06**

a) Describe CMMI technique.
(*Description-1 Mark; Each level -1 Mark)*

Ans: The grading scheme determines compliance with a capability maturity model (CMM) that defines key activities required at different levels of process maturity. The SEI approach provides a measure of the global effectiveness of a company's software engineering practices and establishes five process maturity levels that are defined in the following manner:

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

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

Level 3: Defined. The software process for both management and engineering activities is documented, standardized, and integrated into an organization wide software process. All projects use a documented and approved version of the organization's process for developing and supporting software. This level includes all characteristics defined for level 2.

Level 4: Managed. Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled using detailed measures. This level includes all characteristics defined for level 3.

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

b) Explain personal software process model.
(PSP description- 3 Marks, Activities – 3 Marks)

Ans: The personal software Process (PSP) emphasizes personal measurements of both the work product that is produced and the resultant quality of the work. In order to change ineffective personal process, an individual must go through phases, each requiring training and careful

instrumentation. In addition PSP makes the practitioner responsible for project planning and empowers to control the quality of software product.

PSP model defines following five frame work activities:

Planning- isolates requirements, develops size and resource estimates. Tests are identified and project schedule is created.

High level design: External specification for each component to be constructed is developed and a component design is created.

High level design review: formal verification methods are applied to uncover errors in the design.

Development: component level design is refined and reviewed. Code is generated, reviewed, compiled and tested.

Post-mortem: Using measures and matrix collected, the effectiveness of the process is determined. They provide guidance for improvement.

5. Attempt any two. Marks 16

a) Describe the steps in developing an activity diagram with an example.
(Steps-2 Marks, Example diagram-4 Marks, Any relevant Example explanation-2 Marks)

Ans: Step 1:

First of all, identify the tasks in the project.

Step 2:

You can add more information to the task boxes, such as who is doing the task and the timeframes. You can add this information inside the box or can add it somewhere near the box.

Step 3:

Now, arrange the boxes in the sequence that they are performed during the project execution. The early tasks will be at the left hand side and the tasks performed at the later part of the project execution will be at the right hand side. The tasks that can be performed in parallel should be kept parallel to each other (vertically).

You may have to adjust the sequence a number of times until you get it right. This is why software is an easy tool for creating activity diagrams.

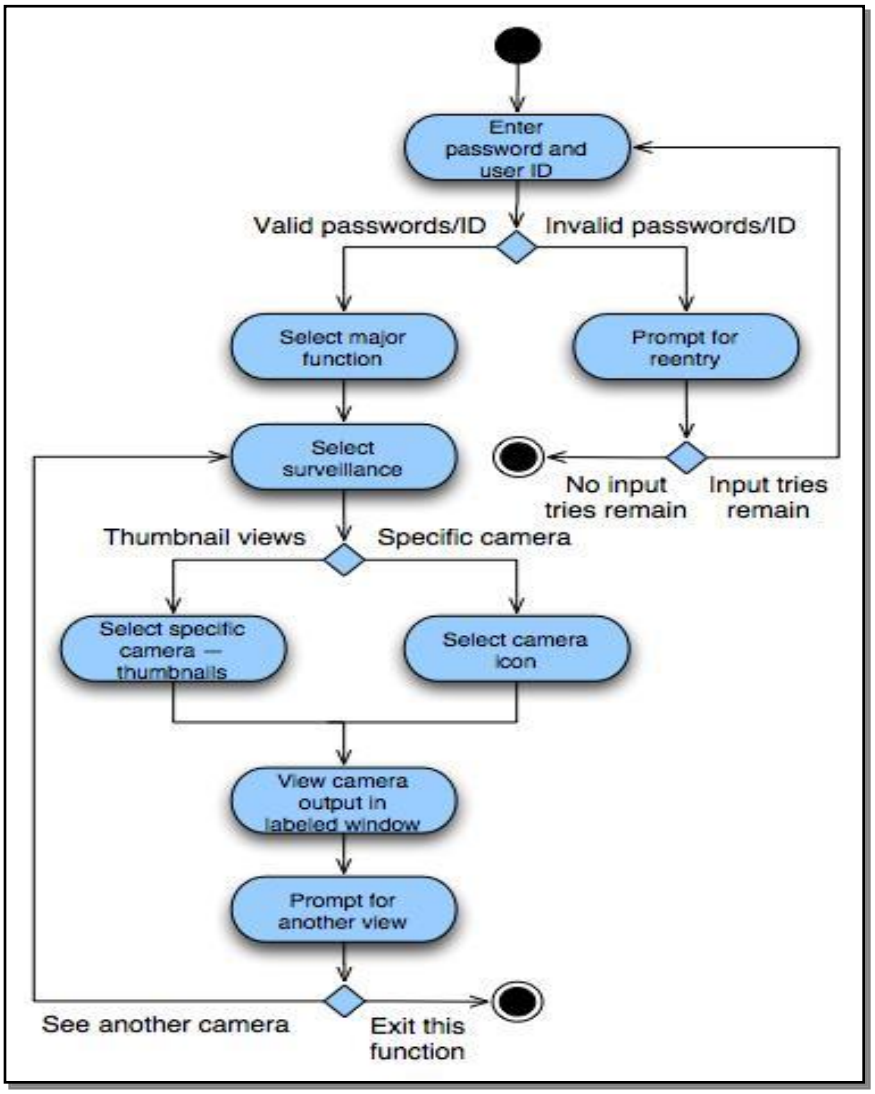
Step 4:

Now, use arrows to join task boxes. These arrows will show the sequence of the tasks. Sometimes, a 'start' and an 'end' box can be added to clearly present the start and the end of the project.

Example of activity diagram:

Activity diagram for Access camera surveillance—display camera views function

The activity diagram is shown for the Function named Access Camera Surveillance- Display Camera Views. The flow of the system is shown in the diagram.



b) Describe project risks, technical risks and business risks.
(Explanation of project and technical risks- 2 ½ Marks each, Explanation of business risk- 3 Marks)

Ans: Project risks:

Project risks threaten the project plan. That is, if project risks become real, it is likely that project schedule will slip and that costs will increase. Project risks identify potential budgetary, schedule, personnel (staffing and organization), resource, customer and requirements problems and their impact on a software project. Project complexity, size, and the degree of structural uncertainty were also defined as project (and estimation) risk factors.

Technical risks:

Technical risks threaten the quality and timeliness of the software to be produced. If a technical risk becomes a reality, implementation may become difficult or impossible. Technical risks identify potential design, implementation, interface, verification, and maintenance problems. In addition, specification ambiguity, technical uncertainty, technical obsolescence, and "leading-edge" technology are also risk factors. Technical risks occur because the problem is harder to solve than we thought it would be.

Business risks:

Business risks threaten the viability of the software to be built. Business risks often jeopardize the project or the product. Candidates for the top five business risks are

(1) Building a excellent product or system that no one really wants (market risk),

(2) Building a product that no longer fits into the overall business strategy for the company (Strategic risk)

(3) Building a product that the sales force doesn't understand how to sell,

(4) Losing the support of senior management due to a change in focus or a change in people (management risk), and

(5) Losing budgetary or personnel commitment (budget risks). It is extremely important to note that simple categorization won't always work. Some risks are simply unpredictable in advance.

c) What is evolutionary process model and describe any one model.

(Explanation of evolutionary model-2 Marks, Description of any one model-6 Marks)

Ans: Evolutionary model:

1. It is the Software, like all complex systems, evolves over a period of time.
2. In this model, Business and product requirements often change as development proceeds, making a straight-line path to an end product is unrealistic.
3. Evolutionary models are iterative.

   There are two evolutionary models:

Prototype Model and Spiral model:

1. Prototype Model:

a. Customers may press for immediate delivery of working but inefficient products

b. The developer often makes implementation compromises in order to get a prototype working quickly.

c. Most helpful when developer is not sure of the efficiency of an algorithm and the adaptability of an operating system.

d. Customer is unaware of detailed input, processing or output requirement.

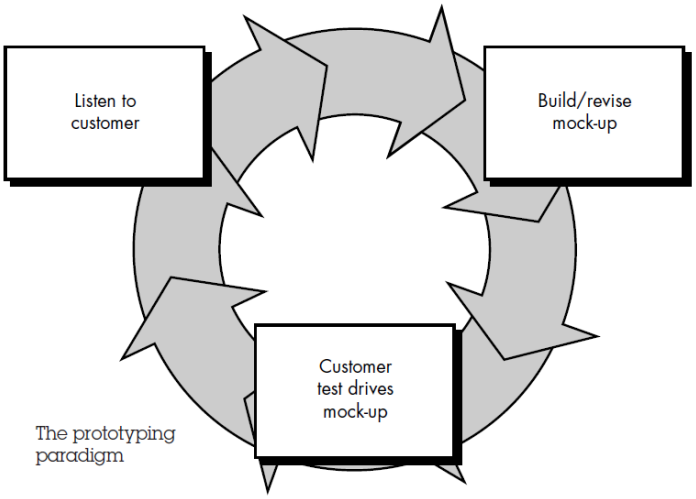
e. Developer and customer determine objectives and draft requirements

f. Prototype quickly produced and evaluated by customer

g. Prototype then refined, and re-evaluated

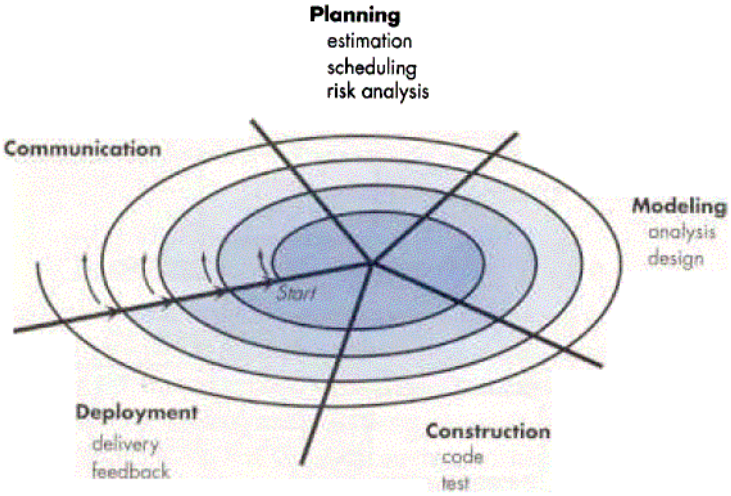
h. Process iterated, before final product development

i. Advantages: Customer participation and better requirements



The prototyping paradigm

2. Spiral model



- Traditional method or model of software development
- Also encompasses all the essential development phases:
  - Requirements analysis
  - Design
  - Code
  - Test
  - Maintenance
- Explicitly addresses the issue of quality assurance by performing the development process in a "step-wise refinement" method
- Each step produces a "deliverable" which embodies the structure or sequential nature of the process
- This process naturally focuses on a high degree of customer or stakeholder involvement during the development process
- In this diagram, "Communication" refers to the Requirements and analysis process, "Planning" corresponds to preliminary design and scheduling, "Modeling" is the detailed

design, "Construction" refers to code/debug/integration, and "Deployment" is the delivery to the customer and the feedback process.

- Software is developed in a series of evolutionary releases.
  - During early iterations, the release might be a paper model or prototype.
  - During later iterations, increasingly more complete versions of the engineered system are produced.
  - The final iteration produces the complete software product.
- First circuit around the spiral might result in the development of the product specification; might result in a review by the customer.
- Next iteration might produce a prototype, containing the GUI, for example; the customer might want to see this, so there could be a PDR and/or CDR at this time.
- Third time around might be used to fill in more detailed functionality, and release a preliminary working model.
- Fourth circuit might result in a complete alpha release, which the customer could "hammer on" for a while to test robustness and provide feedback to the solution provider about the product's strengths and weaknesses.
- Fifth iteration might be a beta test, or it could be the final build for initial release (if the previous circuit was satisfactory enough to warrant this)
- Problem with the spiral model: may not appear controllable to the customer, particularly if the customer is more accustomed to the waterfall model.

6. Attempt any four. Marks 16

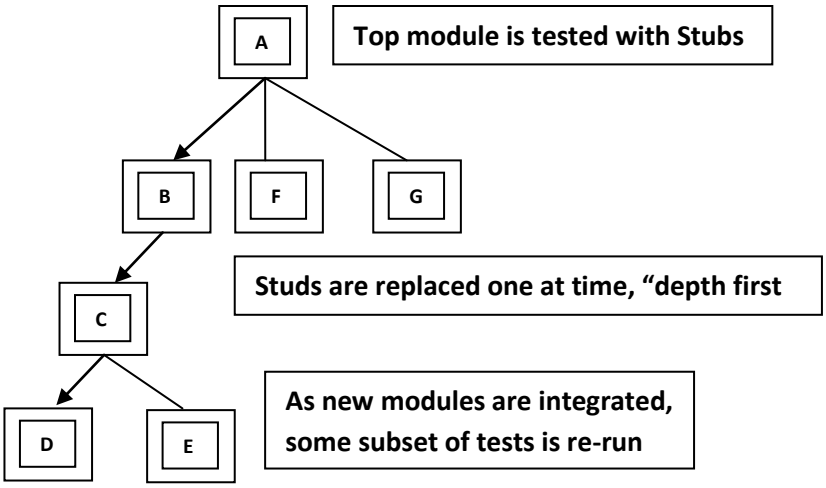
a) Describe Top-down integration testing.

(Explanation- 2 Marks, Diagram-2 Marks)

Ans: Top-down approach:

- In this approach the testing is done in top to down direction.
- The main module is tested first and then the stub are replaced with the original module and then integrated with the main module and then tested.
- Same process is repeated again until all stubs are replaced with original modules and they are completely tested.

Top down Integration



b) Describe with an example generic and customized software.

(Generic software – 1 Mark, Example – 1 Mark, customized software – 1 Mark, Example – 1 Mark)

Ans: Generic software:

1. Generic software is made for general purpose audience.
2. Generic software is tough to make not by skills needed but because of design and need in the market.
3. The software is made by imagining the end-user where end user is just the imagination.
4. This is a class of software that is can be used for a number of different purposes without requiring modification.
5. For example a spreadsheet application is generic because the off-the-shelf package can serve an accountant for finance, engineers for calculation, modeling for forecasting and so on without any modification.

Customized software:

1. Customized software is made to satisfy particular need of particular client.
2. The software is made which has end user face.
3. It is easy to make software because developer knows who is the end user and how to satisfy him.

4. Custom or 'Bespoke' software, is software which has been made for a specific job that software off the shelf could not do.
5. Such software is usually made for a specific company to meet their requirements. It is not usually made to be sold on the mass market.

c) Describe the process used for assessing Risk Impact.
(Each Point-1 Mark (Any Four))

Ans: Three factors affect the consequences that are likely if a risk does occur: its nature, its scope, and its timing.

The nature of the risk indicates the problems that are likely if it occurs. For example, a poorly defined external interface to customer hardware (a technical risk) will preclude early design and testing and will likely lead to system integration problems late in a project.

The scope of a risk combines the severity (just how serious is it?) with its overall distribution (how much of the project will be affected or how many customers are harmed?).

Finally, the timing of a risk considers when and for how long the impact will be felt. In most cases, a project manager might want the "bad news" to occur as soon as possible, but in some cases, the longer the delay, the better.

The following steps are recommended to determine the overall consequences of a risk:

1. Determine the average probability of occurrence value for each risk component.

2. Determine the impact for each component based on the criteria .

3. Complete the risk table and analyze the results as described in the preceding sections.

The overall risk exposure, RE, is determined using the following relationship

RE = P x Cwhere P is the probability of occurrence for a risk, and C is the the cost to the project should the risk occur.

At this point in the risk management process, we have established a set of triplets of

the form:

[ri, li, xi]

whereriis risk, li is the likelihood (probability) of the risk, and xi is the impact of the risk. During risk assessment, we further examine the accuracy of the estimates that were made during risk projection, attempt to rank the risks that have been uncovered, and begin thinking about ways to control and/or avert risks that are likely to occur.

During risk assessment, we perform the following Steps:

1. Define the risk referent levels for the project.

2. Attempt to develop a relationship between each (ri, li, xi) and each of the referent levels.

3. Predict the set of referent points that define a region of termination, bounded by a curve or areas of uncertainty.

4. Try to predict how compound combinations of risks will affect a referent level.

d) State and give the meaning of four McCall's quality factors.
(Any four factors-1 Mark each)

Ans: McCall's Factors:

- **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.
- **Usability:** Effort required to learn, operate, prepare input, and interpret output of a program.
- **Maintainability:** Effort required to locate and fix an error in a program. [This is a very limited definition.]
- **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.

e) Justify why testing is also termed as a quality control measure.
(Each Point-1 Mark (Any Four))

(Any relevant description shall be considered)

Ans:

- **Quality Control** describes the directed use of testing to measure the achievement of a specified standard. Quality control is a use of testing. Quality control is a superset of testing, although it often used synonymously with testing. Roughly, you test to see if something is broken, and with quality control you set limits that say, in effect, if this particular stuff is broken then whatever you're testing *fails*.
- For quality control to be effective, you must test the same things the same way every time you test. When you change your tests, your results become inconsistent. You need a test plan.
- A **test plan** is simply a high-level summary of the areas (functionality, elements, regions, etc.) You will test, how frequently you will test them, and where in the development or publication process you will test them. A test plan also needs an estimate of the duration of testing, and statement of any required resources.
- The major phases of a web site need test plans, because the focus and emphasis of testing will change over time. Testing a new site in development is very different from testing a site that has been up and running for some time. Furthermore, any changes to the website code, incremental of major, require regression test plans.