Code No. 3434 /N/S

FACULTY OF ENGINEERING

B.E. 3/4 (CSE) I-Semester (New) (Suppl.) Examination, May / June 2017

Subject: Software Engineering

Time: 3 hours **Max. Marks: 75**

Note: Answer all questions from Part-A. Answer any FIVE questions Part-B

PART – A (25 Marks)

1 Why software engineering is said to be layered technology? 2

2 What is Agility? 3

3 What is an analysis model? 2

4 Briefly discuss about the tracking the progress of the project. 3

5 Define cardinality and modality. 2

6 Distinguish between refactoring and refinement. 3

7 What is software architecture? 2

8 What is object constraint language? 3

9 What is stress testing? 2

10 What is pattern? Discuss about testing patterns. 3

PART – B (50 Marks)

C a) What is process framework? Explain about the umbrella activities of a software process framework. 5

b) Distinguish between incremental and evolutionary process models. Explain spiral model in detail. 5

12 a) Explain about software project personnel. 5

b) What is the purpose of requirements Elicitation? Who are the different stake Holders involved in requirements elicitation? 5

13 a) Enumerate the characteristics of a good software design. 5

b) Explain Scenario based modeling approach. 5

14 a) Define cohesion and coupling. Explain its types. 5

b) What is a component? Explain how to conduct component-level design. 5

15 a) Define software testing. Explain Alpha testing and Beta testing. 5

b) Explain the concept of basis path testing in detail with an example. 5

16 a) Explain O-O testing methods. 5

b) List out ISO quality factors and discuss about the metrics for maintenance. 5

17 Write short notes on the following :

a) Unified process 4

b) Regression testing 3

c) Personal and team process 3

******

1. Why software engineering is said to be layered technology?

Ans. Software engineering is a fully layered technology.

- To develop a software, we need to go from one layer to another.
- All these layers are related to each other and each layer demands the fulfilment of the previous layer.

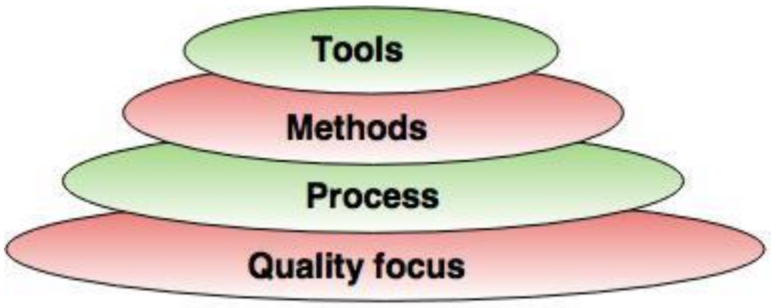


Fig. - Software Engineering Layers

- Software should achieve a good quality in design and meet all the specifications of the customer.
- Software does not wear out i.e. it does not lose the material.
- Software should be inherently complex.
- Software must be efficient i.e. the ability of the software to use system resources in an effective and efficient manner.
- Software must be integral i.e. it must prevent from unauthorized access to the software or data.

2 What is Agility?

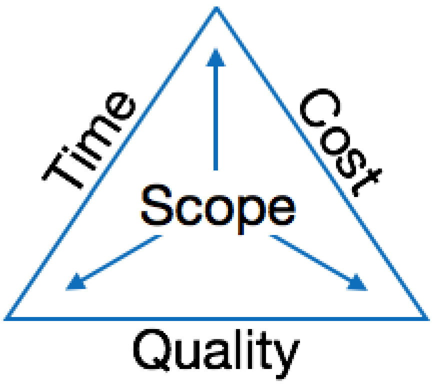
Ans. Agility means effective (rapid and adaptive) response to change, effective communication among all stockholder. Drawing the customer onto team and organizing a team so that it is in control of work performed. ... The agile process forces the development team to focus on software itself rather than design and documentation.

3 What is an analysis model?

Ans. Analysis Model is a technical representation of the system. It acts as a link between system description and design model. In Analysis Modelling, information, behavior and functions of the system is defined and translated into the architecture, component and interface level design in the design modeling.

4 Briefly discuss about the tracking the progress of the project.

Ans.



- Every project may have a unique and distinct goal.
- Project is not routine activity or day-to-day operations.
- Project comes with a start time and end time.
- Project ends when its goal is achieved hence it is a temporary phase in the lifetime of an organization.
- Project needs adequate resources in terms of time, manpower, finance, material and knowledge-bank.

A project manager closely monitors the development process, prepares and executes various plans, arranges necessary and adequate resources, maintains communication among all team members in order to address issues of cost, budget, resources, time, quality and customer satisfaction.

5 Define cardinality and modality.

Ans. The cardinality and modality are the concepts mainly used while designing a database and holds great importance. The cardinality describes the number of relations an object possesses for another object. On the other hand, modality describes whether the relation is required to be made or not.

6 Distinguish between refactoring and refinement.

Ans. Model refactoring is a transformation used to improve the structure of a model while preserving its behavior. Model refinement is a transformation that adds more detail to an existing model**.**

7 What is software architecture?

Ans. Software architecture is, simply, the organization of a system. This organization includes all components, how they interact with each other, the environment in which they operate, and the principles used to design the software. In many cases, it can also include the evolution of the software into the future.

8 What is object constraint language?

Ans. Object Constraint Language (OCL), is a formal language to express side effect-free constraints. Users of the Unified Modeling Language and other languages can use OCL to specify constraints and other expressions attached to their models.

9 What is stress testing?

Ans. Stress testing is a software testing activity that determines the robustness of software by testing beyond the limits of normal operation. Stress testing is particularly important for "mission critical" software but is used for all types of software.

10 What is pattern? Discuss about testing patterns.

Ans. Software designers adapt the pattern solution to their specific project. Patterns use a formal approach to describe a design problem, its proposed solution, and any other factors that might affect the problem or the solution.

A pattern can contain the description of certain objects and object classes to be used, along with their attributes and dependencies, and the general approach to how to solve the problem. Often, programmers can use more than one pattern to address a specific problem. A collection of patterns is called a pattern framework.

11 a) What is process framework? Explain about the umbrella activities of a software process framework.

b) Distinguish between incremental and evolutionary process models. Explain Spiral model in detail.

Ans. A process framework establishes the foundation for a complete software process by identifying a small number of framework activities that are applicable to all software projects, regardless of size or complexity. It also includes a set of umbrella activities that are applicable across the entire software process.

Typical umbrella activities are:

1. Software project tracking and control

- In this activity, the developing team accesses project plan and compares it with the predefined schedule.
- If these project plans do not match with the predefined schedule, then the required actions are taken to maintain the schedule.

2. Risk management

- Risk is an event that may or may not occur.
- If the event occurs, then it causes some unwanted outcome. Hence, proper risk management is required.

3. Software Quality Assurance (SQA)

- SQA is the planned and systematic pattern of activities which are required to give a guarantee of software quality.

  For example, during the software development meetings are conducted at every stage of development to find out the defects and suggest improvements to produce good quality software.

4. Formal Technical Reviews (FTR)

- FTR is a meeting conducted by the technical staff.
- The motive of the meeting is to detect quality problems and suggest improvements.
- The technical person focuses on the quality of the software from the customer point of view.

5. Measurement

- Measurement consists of the effort required to measure the software.
- The software cannot be measured directly. It is measured by direct and indirect measures.
- Direct measures like cost, lines of code, size of software etc.
- Indirect measures such as quality of software which is measured by some other factor. Hence, it is an indirect measure of software.

6. Software Configuration Management (SCM)

- It manages the effect of change throughout the software process.

7. Reusability management

- It defines the criteria for reuse the product.
- The quality of software is good when the components of the software are developed for certain application and are useful for developing other applications.

8. Work product preparation and production

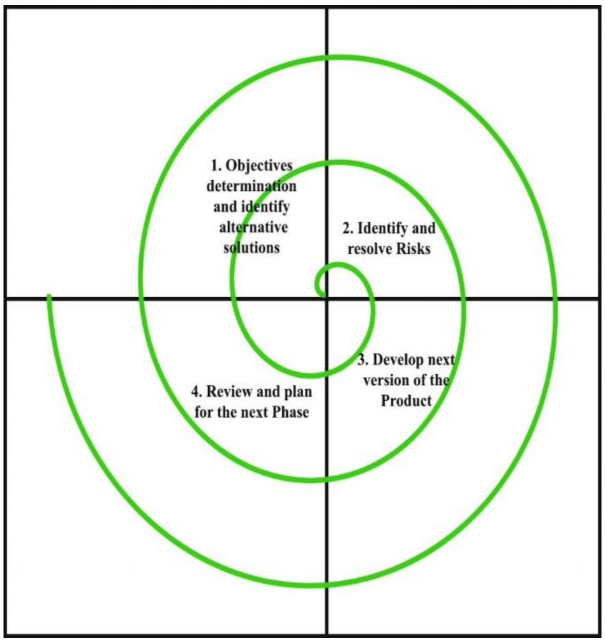
- It consists of the activities that are needed to create the documents, forms, lists, logs and user manuals for developing a software.

B) The Evolutionary model, the complete cycle of activities is repeated for each version. In the Incremental model, increments are individually designed, tested, and delivered at successive points in time. In the high-risk model, the project is divided into phases and each phase helps constrain risk.

Incremental Model is a process of software development where requirements divided into multiple standalone modules of the software development cycle. In this model, each module goes through the requirements, design, implementation and testing phases. Every subsequent release of the module adds function to the previous release. The process continues until the complete system achieved.

Evolutionary model is a combination of Iterative and Incremental model of software development life cycle. The Evolutionary development model divides the development cycle into smaller, incremental waterfall models in which users are able to get access to the product at the end of each cycle.

Spiral model is one of the most important Software Development Life Cycle models, which provides support for Risk Handling. In its diagrammatic representation, it looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. Each loop of the spiral is called a Phase of the software development process. The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks. As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using the spiral model.
The Radius of the spiral at any point represents the expenses(cost) of the project so far, and the angular dimension represents the progress made so far in the current phase.


1. Objectives determination and identify alternative solutions
2. Identify and resolve Risks
3. Develop next version of the Product
4. Review and plan for the next Phase

1. **Objectives determination and identify alternative solutions:** Requirements are gathered from the customers and the objectives are identified, elaborated, and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.

2. **Identify and resolve Risks:** During the second quadrant, all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution are identified and the risks are resolved using the best possible strategy. At the end of this quadrant, the Prototype is built for the best possible solution.

3. **Develop next version of the Product:** During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.

4. **Review and plan for the next Phase:** In the fourth quadrant, the Customers evaluate the so far developed version of the software. In the end, planning for the next phase is started.

Risk Handling in Spiral Model
A risk is any adverse situation that might affect the successful completion of a software project. The most important feature of the spiral model is handling these unknown risks

after the project has started. Such risk resolutions are easier done by developing a prototype. The spiral model supports coping up with risks by providing the scope to build a prototype at every phase of the software development.
Advantages of Spiral Model:

1. **Risk Handling:** The projects with many unknown risks that occur as the development proceeds, in that case, Spiral Model is the best development model to follow due to the risk analysis and risk handling at every phase.

2. **Good for large projects:** It is recommended to use the Spiral Model in large and complex projects.

3. **Flexibility in Requirements:** Change requests in the Requirements at later phase can be incorporated accurately by using this model.

4. **Customer Satisfaction:** Customer can see the development of the product at the early phase of the software development and thus, they habituated with the system by using it before completion of the total product.

Disadvantages of Spiral Model:

1. **Complex:** The Spiral Model is much more complex than other SDLC models.

2. **Expensive:** Spiral Model is not suitable for small projects as it is expensive.

3. **Too much dependability on Risk Analysis:** The successful completion of the project is very much dependent on Risk Analysis. Without very highly experienced experts, it is going to be a failure to develop a project using this model.

4. **Difficulty in time management:** As the number of phases is unknown at the start of the project, so time estimation is very difficult.

12 a) Explain about software project personnel.

Ans. It involves:

- o Defining requirement for personnel
- o Recruiting (identifying, interviewing, and selecting candidates)
- o Compensating
- o Developing and promoting agent

For personnel planning and scheduling, it is helpful to have efforts and schedule size for the subsystems and necessary component in the system.

At planning time, when the system method has not been completed, the planner can only think to know about the large subsystems in the system and possibly the major modules in these subsystems.

Once the project plan is estimated, and the effort and schedule of various phases and functions are known, staff requirements can be achieved.

From the cost and overall duration of the projects, the average staff size for the projects can be determined by dividing the total efforts (in person-months) by the whole project duration (in months).

Lead and manage the project team. Determine the methodology used on the project. Establish a project schedule and determine each phase. Assign tasks to project team members.

Typically, the staff required for the project is small during requirement and design, the maximum during implementation and testing, and drops again during the last stage of integration and testing.

Using the COCOMO model, average staff requirement for various phases can be calculated as the effort and schedule for each method are known.

When the schedule and average staff level for every action are well-known, the overall personnel allocation for the project can be planned.

This plan will indicate how many people will be required for different activities at different times for the duration of the project.

The total effort for each month and the total effort for each step can easily be calculated from this plan.

B. What is the purpose of requirements Elicitation? Who are the different stake

Holders involved in requirements elicitation?

Ans. Requirements elicitation is perhaps the most difficult, most error-prone and most communication intensive software development. It can be successful only through an effective customer-developer partnership. It is needed to know what the users really need.

Requirements elicitation Activities:

Requirements elicitation includes the subsequent activities. Few of them are listed below –

- Knowledge of the overall area where the systems is applied.
- The details of the precise customer problem where the system are going to be applied must be understood.
- Interaction of system with external requirements.
- Detailed investigation of user needs.
- Define the constraints for system development.

The requirements definition process begins with the elicitation of stakeholder requirements, the first step of which is to identify the stakeholders from whom those requirements are to be gathered. It is common in requirements engineering to define a stakeholder as someone who has a stake in the project—that is, someone who is affected by the system in some way, or can affect the system in some way. In most systems, this definition is not useful since, regardless of where the system boundary is set, it is often difficult to find someone who is not affected by the system. While we must take into account anyone or anything that is affected by, or that can affect, the system when considering requirements, such entities are not automatically (nor necessarily) stakeholders. This paper proposes that a more useful definition of a stakeholder is someone who has a right to influence the system. A method for selecting stakeholders is proposed, and a simple example is provided to illustrate the process.

13 a) Enumerate the characteristics of a good software design.

b) Explain Scenario based modeling approach.

Ans. Characteristics that a good software design

1. Correctness
2. Understandability
3. Efficiency
4. Maintainability

1) Correctness

The design of any software is evaluated for its correctness. The evaluators check the software for every kind of input and action and observe the results that the software will produce according to the proposed design. If the results are correct for every input, the design is accepted and is considered that the software produced according to this design will function correctly.

2) Understandability

The software design should be understandable so that the developers do not find any difficulty to understand it. Good software design should be self- explanatory. This is because there are hundreds and thousands of developers that develop different modules of the software, and it would be very time consuming to explain each design to each developer. So, if the design is easy and self- explanatory, it would be easy for the developers to implement it and build the same software that is represented in the design.

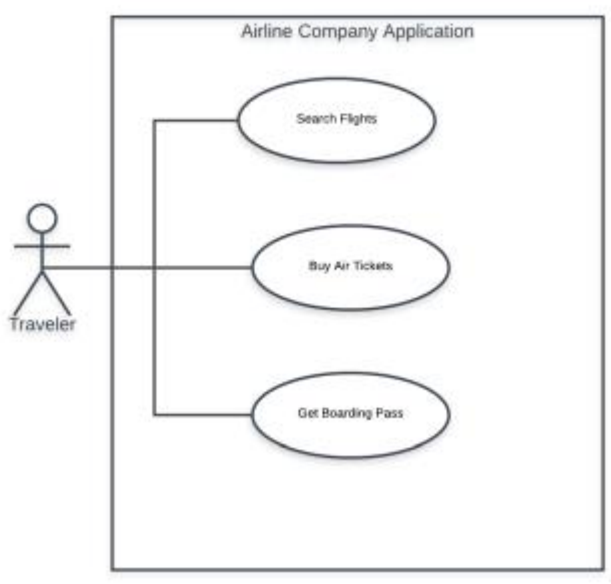
3) Efficiency

The software design must be efficient. The efficiency of the software can be estimated from the design phase itself, because if the design is describing software that is not efficient and useful, then the developed software would also stand on the same level of efficiency. Hence, for efficient and good quality software to be developed, care must be taken in the designing phase itself.

4) Maintainability

The software design must be in such a way that modifications can be easily made in it. This is because every software needs time to time modifications and maintenance. So, the design of the software must also be able to bear such changes. It should not be the case that after making some modifications the other features of the software start misbehaving. Any change made in the software design must not affect the other available features, and if the features are getting affected, then they must be handled properly.

B. Requirements modeling is the process of identifying the requirements this software solution must meet in order to be successful. Requirements modeling contains several sub-stages, typically: scenario-based modeling, flow-oriented modeling, data modeling, class-based modeling, and behavioral modeling. Also, as the term "modeling" implies, all of these stages typically result in producing diagrams that visually convey the concepts they identify. The most common method for creating these diagrams is Unified Modeling Language (UML).



Scenario-based modeling is one of the sub-stages of requirements modeling. It's also typically the first stage of requirements modeling, since it identifies the primary use cases for the proposed software system or application, to which later stages of requirements modeling.

The use case is essentially a primary example of how the proposed software application or system is meant to be used, from the user's point of view. A use case diagram will typically show system actors, humans or other entities external to the system and how they interact with the system. Technically, each action such a system actor can perform with the application or system is considered to be a separate use case.

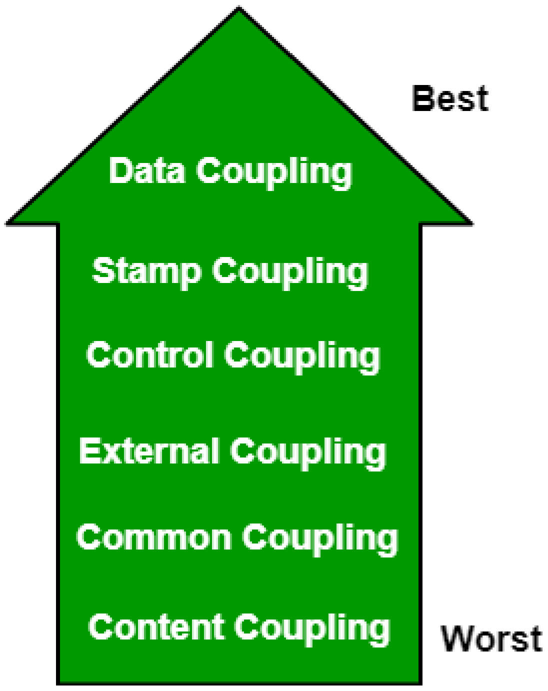
The example use case diagram here, drawn using UML, depicts three possible use cases for the system actor "Traveler" when interacting with an airline company's application. The three use case actions depicted are:

1. Search for flights
2. Buy tickets
3. Get boarding pass

14 a) Define cohesion and coupling. Explain its types.

b) What is a component? Explain how to conduct component-level design.

Ans. **Coupling:** Coupling is the measure of the degree of interdependence between the modules. A good software will have low coupling.

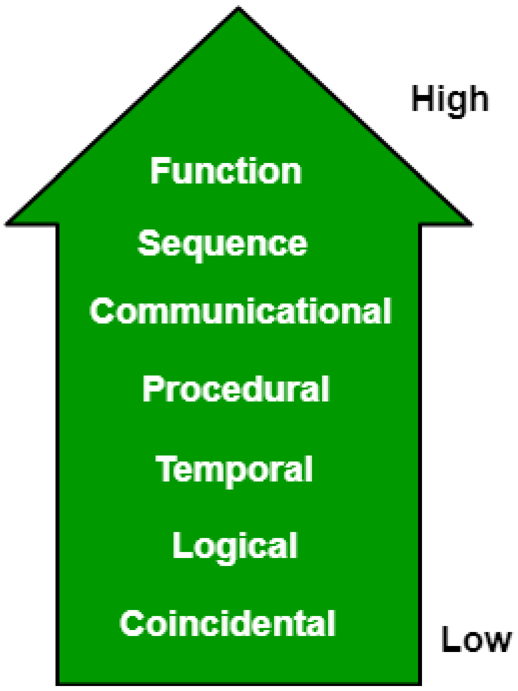


Types of Coupling:

- **Data Coupling:** If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled. In data coupling, the components are independent to each other and communicating through data. Module communications don't contain tramp data. Example-customer billing system.
- **Stamp Coupling** In stamp coupling, the complete data structure is passed from one module to another module. Therefore, it involves tramp data. It may be necessary due to efficiency factors- this choice made by the insightful designer, not a lazy programmer.
- **Control Coupling:** If the modules communicate by passing control information, then they are said to be control coupled. It can be bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality. Example- sort function that takes comparison function as an argument.
- **External Coupling:** In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware. Ex- protocol, external file, device format, etc.
- **Common Coupling:** The modules have shared data such as global data structures. The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change. So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses and reduced maintainability.

- **Content Coupling:** In a content coupling, one module can modify the data of another module or control flow is passed from one module to the other module. This is the worst form of coupling and should be avoided.

Cohesion: Cohesion is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Basically, cohesion is the internal glue that keeps the module together. A good software design will have high cohesion.



Types of Cohesion:

- **Functional Cohesion:** Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. It is an ideal situation.
- **Sequential Cohesion:** An element outputs some data that becomes the input for other element, i.e., data flow between the parts. It occurs naturally in functional programming languages.
- **Communicational Cohesion:** Two elements operate on the same input data or contribute towards the same output data. Example- update record in the database and send it to the printer.
- **Procedural Cohesion:** Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable. Ex- calculate student GPA, print student record, calculate cumulative GPA, print cumulative GPA.
- **Temporal Cohesion:** The elements are related by their timing involved. A module connected with temporal cohesion all the tasks must be executed in the same time-span. This cohesion contains the code for initializing all the parts of the system. Lots of different activities occur, all at unit time.
- **Logical Cohesion:** The elements are logically related and not functionally. Ex- A component reads inputs from tape, disk, and network. All the code for these functions is

in the same component. Operations are related, but the functions are significantly different.

- **Coincidental Cohesion:** The elements are not related(unrelated). The elements have no conceptual relationship other than location in source code. It is accidental and the worst form of cohesion. Ex- print next line and reverse the characters of a string in a single component.

B) What is a component? Explain how to conduct component-level design.

Ans. A component is a modular, portable, replaceable, and reusable set of well-defined functionality that encapsulates its implementation and exporting it as a higher-level interface.

A component is a software object, intended to interact with other components, encapsulating certain functionality or a set of functionalities. It has an obviously defined interface and conforms to a recommended behaviour common to all components within an architecture.

A software component can be defined as a unit of composition with a contractually specified interface and explicit context dependencies only. That is, a software component can be deployed independently and is subject to composition by third parties.

Conducting Component-Level Design

Recognizes all design classes that correspond to the problem domain as defined in the analysis model and architectural model.

- Recognizes all design classes that correspond to the infrastructure domain.
- Describes all design classes that are not acquired as reusable components, and specifies message details.
- Identifies appropriate interfaces for each component and elaborates attributes and defines data types and data structures required to implement them.
- Describes processing flow within each operation in detail by means of pseudo code or UML activity diagrams.
- Describes persistent data sources (databases and files) and identifies the classes required to manage them.
- Develop and elaborates behavioural representations for a class or component. This can be done by elaborating the UML state diagrams created for the analysis model and by examining all use cases that are relevant to the design class.
- Elaborates deployment diagrams to provide additional implementation detail.
- Demonstrates the location of key packages or classes of components in a system by using class instances and designating specific hardware and operating system environment.
- The final decision can be made by using established design principles and guidelines. Experienced designers consider all (or most) of the alternative design solutions before settling on the final design model.

15 a) Define software testing. Explain Alpha testing and Beta testing.

b) Explain the concept of basis path testing in detail with an example.

Ans. Software testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is Defect free. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest. The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.

What Is Alpha Testing?
This is a form of internal acceptance testing performed mainly by the in-house software QA and testing teams. Alpha testing is the last testing done by the test teams at the development site after the acceptance testing and before releasing the software for the beta test.

Alpha testing can also be done by the potential users or customers of the application. But still, this is a form of in-house acceptance testing.

What Is Beta Testing?
This is a testing stage followed by the internal full alpha test cycle. This is the final testing phase where the companies release the software to a few external user groups outside the company test teams or employees. This initial software version is known as the beta version. Most companies gather user feedback in this release.

Alpha Testing	Beta Testing
Alpha testing involves both the white box and black box testing.	Beta testing commonly uses black box testing.
Alpha testing is performed by testers who are usually internal employees of the organization.	Beta testing is performed by clients who are not part of the organization.
Alpha testing is performed at developer's site.	Beta testing is performed at end-user of the product.
Reliability and security testing are not checked in alpha testing.	Reliability, security and robustness are checked during beta testing.
Alpha testing ensures the quality of the product before forwarding to beta testing.	Beta testing also concentrates on the quality of the product but collects users input on the product and ensures that the product is ready for real time users.

Alpha Testing	Beta Testing
Alpha testing requires a testing environment or a lab.	Beta testing doesn’t require a testing environment or lab.
Alpha testing may require long execution cycle.	Beta testing requires only a few weeks of execution.
Developers can immediately address the critical issues or fixes in alpha testing.	Most of the issues or feedback collected from beta testing will be implemented in future versions of the product.

B) **Basis Path Testing** is a white-box testing technique based on the control structure of a program or a module. Using this structure, a control flow graph is prepared and the various possible paths present in the graph are executed as a part of testing. Therefore, by definition,
Basis path testing is a technique of selecting the paths in the control flow graph that provide a basis set of execution paths through the program or module.
Since this testing is based on the control structure of the program, it requires complete knowledge of the program’s structure. To design test cases using this technique, four steps are followed:

1. Construct the Control Flow Graph
2. Compute the Cyclomatic Complexity of the Graph
3. Identify the Independent Paths
4. Design Test cases from Independent Paths

16 a) Explain O-O testing methods.

b) List out ISO quality factors and discuss about the metrics for maintenance.

Ans. Object Oriented Testing methods:

Testing is a continuous activity during software development. In object-oriented systems, testing encompasses three levels, namely, unit testing, subsystem testing, and system testing.

Unit Testing:

- In unit testing, the individual classes are tested. It is seen whether the class attributes are implemented as per design and whether the methods and the interfaces are error-free.
- Unit testing is the responsibility of the application engineer who implements the structure.

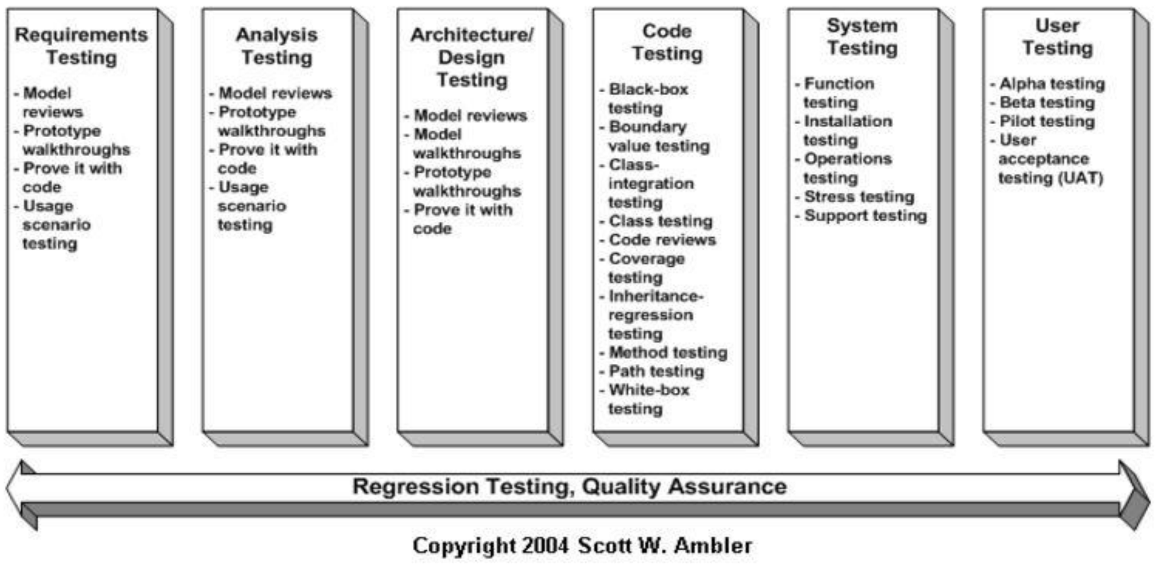
Subsystem Testing:

- This involves testing a particular module or a subsystem and is the responsibility of the subsystem lead. It involves testing the associations within the subsystem as well as the interaction of the subsystem with the outside.

- Subsystem tests can be used as regression tests for each newly released version of the subsystem.

System Testing:

- System testing involves testing the system as a whole and is the responsibility of the quality-assurance team. The team often uses system tests as regression tests when assembling new
  releases.



Object-Oriented Testing Techniques:

Grey Box Testing:

The different types of test cases that can be designed for testing object-oriented programs are called grey box test cases. Some of the important types of grey box testing are:

- **State model based testing:** This encompasses state coverage, state transition coverage, and state transition path coverage.
- **Use case based testing:** Each scenario in each use case is tested.
- **Class diagram based testing:** Each class, derived class, associations, and aggregations are tested.
- **Sequence diagram based testing:** The methods in the messages in the sequence diagrams are tested.

Techniques for Subsystem Testing:

The two main approaches of subsystem testing are:

- **Thread based testing:** All classes that are needed to realize a single use case in a subsystem are integrated and tested.
- **Use based testing:** The interfaces and services of the modules at each level of hierarchy are tested. Testing starts from the individual classes to the small modules comprising of classes, gradually to larger modules, and finally all the major subsystems.

Categories of System Testing:

- **Alpha testing:** This is carried out by the testing team within the organization that develops software.
- **Beta testing:** This is carried out by select group of co-operating customers.
- **Acceptance testing:** This is carried out by the customer before accepting the deliverables.

Black-box testing:
Testing that verifies the item being tested when given the appropriate input provides the expected results.

Boundary-value testing:

Testing of unusual or extreme situations that an item should be able to handle.

Class testing:
The act of ensuring that a class and its instances (objects) perform as defined.

Component testing:
The act of validating that a component works as defined.

Inheritance-regression testing:

The act of running the test cases of the super classes, both direct and indirect, on a given subclass.

Integration testing:
Testing to verify several portions of software work together.

Model review:
An inspection, ranging anywhere from a formal technical review to an informal walkthrough, by others who were not directly involved with the development of the model.

Path testing:
The act of ensuring that all logic paths within your code are exercised at least once.

Regression testing:

The acts of ensuring that previously tested behaviors still work as expected after changes have been made to an application.

Stress testing:

The act of ensuring that the system performs as expected under high volumes of transactions, users, load, and so on.

Technical review:

A quality assurance technique in which the design of your application is examined critically by a group of your peers. A review typically focuses on accuracy, quality, usability, and completeness. This process is often referred to as a walkthrough, an inspection, or a peer review.

User interface testing:

The testing of the user interface (UI) to ensure that it follows accepted UI standards and meets the requirements defined for it. Often referred to as graphical user interface (GUI) testing.

White-box testing:
Testing to verify that specific lines of code work as defined. Also referred to as clear-box testing.

B) List out ISO quality factors and discuss about the metrics for maintenance.

Ans.

Reliability

Reliability requirements deal with service failure. They determine the maximum allowed failure rate of the software system, and can refer to the entire system or to one or more of its separate functions.

Efficiency

It deals with the hardware resources needed to perform the different functions of the software system. It includes processing capabilities (given in MHz), its storage capacity (given in MB or GB) and the data communication capability (given in MBPS or GBPS).

It also deals with the time between recharging of the system's portable units, such as, information system units located in portable computers, or meteorological units placed outdoors.

Integrity

This factor deals with the software system security, that is, to prevent access to unauthorized persons, also to distinguish between the group of people to be given read as well as write permit.

Usability

Usability requirements deal with the staff resources needed to train a new employee and to operate the software system.

Maintainability

This factor considers the efforts that will be needed by users and maintenance personnel to identify the reasons for software failures, to correct the failures, and to verify the success of the corrections.

Flexibility

This factor deals with the capabilities and efforts required to support adaptive maintenance activities of the software. These include adapting the current software to additional circumstances and customers without changing the software. This factor's requirements also support perfective maintenance activities, such as changes and additions to the software in order to improve its service and to adapt it to changes in the firm's technical or commercial environment.

Testability

Testability requirements deal with the testing of the software system as well as with its operation. It includes predefined intermediate results, log files, and also the automatic diagnostics performed by the software system prior to starting the system, to find out whether all components of the system are in working order and to obtain a report about the detected faults. Another type of these requirements deals with automatic diagnostic checks applied by the maintenance technicians to detect the causes of software failures.

17 Write short notes on the following :

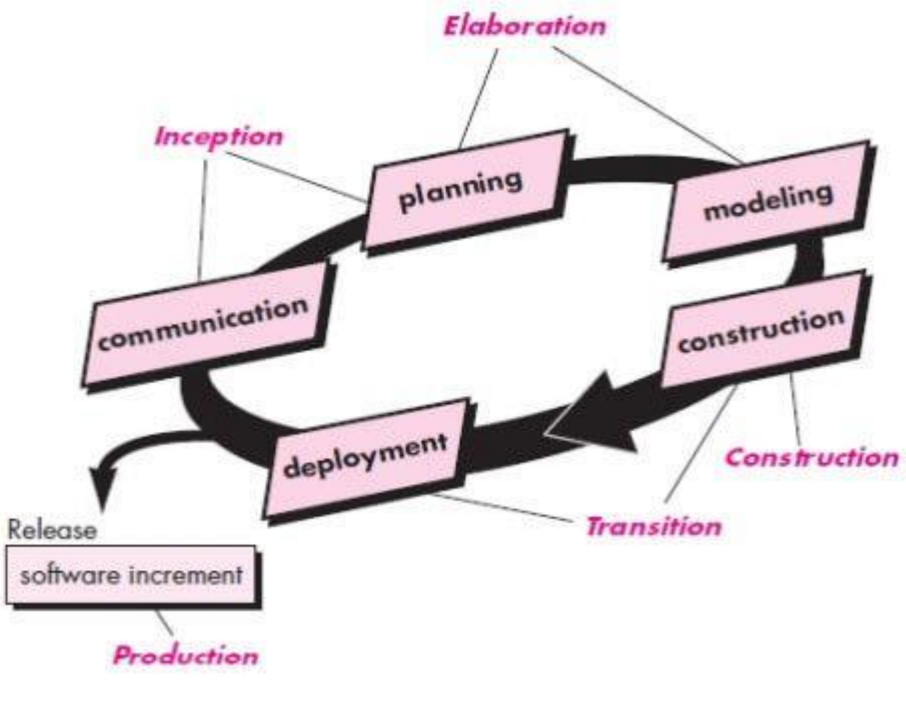
a) Unified process

b) Regression testing

c) Personal and team process

Ans.

a) Unified process



Unified process (UP) is an architecture centric, use case driven, iterative and incremental development process. UP is also referred to as the unified software development process.

The Unified Process is an attempt to draw on the best features and characteristics of traditional software process models, but characterize them in a way that implements many of the best principles of agile software development. The Unified Process recognizes the importance of customer communication and streamlined methods for describing the customer's view of a system. It emphasizes the important role of software architecture and "helps the architect focus on the right goals, such as understand ability, reliance to future changes, and reuse" . It suggests a process flow that is iterative and incremental, providing the evolutionary feel that is essential in modern software development.

Phases of the Unified Process

This process divides the development process into five phases:

- Inception
- Elaboration
- Conception
- Transition
- Production

The inception phase of the UP encompasses both customer communication and planning activities. By collaborating with stakeholders, business requirements for the software are identified; a rough architecture for the system is proposed; and a plan for the iterative, incremental nature of the ensuing project is developed.

The elaboration phase encompasses the communication and modeling activities of the generic process model. Elaboration refines and expands the preliminary use cases that were developed as part of the inception phase and expands the architectural representation to include five different views of the software the use case model, the requirements model, the design model, the implementation model, and the deployment model. Elaboration creates an "executable architectural baseline" that represents a "first cut" executable system.

The construction phase of the UP is identical to the construction activity defined for the generic software process. Using the architectural model as input, the construction phase develops or acquires the software components that will make each use case operational for end users.

The transition phase of the UP encompasses the latter stages of the generic construction activity and the first part of the generic deployment (delivery and feedback) activity. Software is given to end users for beta testing and user feedback reports both defects and necessary changes.

The production phase of the UP coincides with the deployment activity of the generic process. During this phase, the ongoing use of the software is monitored, support for the operating environment (infrastructure) is provided, and defect reports and requests for changes are submitted and evaluated. It is likely that at the same time the construction, transition, and production phases are being conducted, work may have already begun on the next software increment. This means that the five UP phases do not occur in a sequence, but rather with staggered concurrency.

B. Regression testing

Regression testing is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features.

Regression Testing is nothing but a full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine.

This testing is done to make sure that new code changes should not have side effects on the existing functionalities. It ensures that the old code still works once the latest code changes are done.

C. Personal and team process

The Personal Software Process (PSP) emphasizes personal measurement of both the work product that is produced and the resultant quality of the work product. In addition PSP makes the practitioner responsible for project planning and empowers the practitioner to control the quality of all software work products that are developed. The PSP model defines five framework activities:

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

- **High level design.** External specifications for each component to be constructed are developed and a component design is created. Prototypes are built when uncertainty exists. All issues are recorded and tracked.
- **High level design review.** Formal verification methods are applied to uncover errors in the design. Metrics are maintained for all important tasks and work results.
- **Development.** The component level design is refined and reviewed. Code is generated, reviewed, compiled, and tested. Metrics are maintained for all important tasks and work results.
- **Postmortem.** Using the measures and metrics collected, the effectiveness of the process is determined. Measures and metrics should provide guidance for modifying the process to improve its effectiveness.