Code No. 11590/CBCS

FACULTY OF ENGINEERING

B.E. VI – Semester (CBCS)(CSE) (Main) Examination, April/May 2019

Subject: Software Engineering

Time : 3 Hours **Max. Mark: 70**

PART – A (20 Marks)

Note: Answer all questions from Part – A & any five questions from Part-B.

1.	What are umbrella activities? Give an example?	2
2.	What is an agile process? List few agile, Process Models	2
3.	What are the principles of modeling?	2
4.	What is the role of requirements engineer in requirements elaboration phase?	2
5.	List the goals of a good design.	2
6.	What is scenario-based modeling?	2
7.	What is the purpose of a Data design?	2
8.	Differentiate between coupling and cohension.	2
9.	What is software quality Assurance (SQA)?	2
10.	Given any 2 differences between Black box & white box testing.	2

PART – B (50 Marks)

11.	a) Briefly explain evolutionary process models?	5
	b) Explain process framework with an example?	5
12.	a) Explain about Business process Engineering?	5
	b) What is requirements engineering? Explain about elicitation in detail.	5
13.	a) Explain in detail about flow oriented modeling.	5
	b) Briefly explain software design process and quality.	5
14.	Explain Architecture styles and patterns in details.	10
15.	a) Explain ISO 9000 quality standards.	5
	b) What is debugging? Explain the process of debugging.	5
16.	a) Explain Object Oriented test strategies.	5
	b) Explain metrics for maintenance.	5
17.	Explain	
	(i) Interface Design Steps	5
	(ii) Validation testing	5

****

1.What are umbrella activities? Give an example?

Ans. The umbrella activities of a software process are:

- Software project tracking and control.
- Risk Management.
- Software Quality Assurance.
- Formal Technical Reviews.
- Software Configuration Management.
- Work product preparation and production.
- Reusability management, Measurement.

2. What is an agile process? List few agile, Process Models

Ans. The meaning of Agile is swift or versatile. **"Agile process model**" refers to a software development approach based on iterative development. Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning. The project scope and requirements are laid down at the beginning of the development process.

Agile Process Models

- Extreme Programming (XP)
- Adaptive Software Development (ASD)
- Dynamic Systems Development Method (DSDM)
- Scrum.
- Crystal.
- Feature Driven Development (FDD)
- Agile Modeling (AM)

3. What are the principles of modeling?

Ans.
Principle 1: Design should be traceable to the requirements model.
Principle 2: Always consider the architecture of the system to be built.
Principle 3: Design of data is as important as design of processing.
Principle 4: Interfaces must be designed.

4.What is the role of requirements engineer in requirements elaboration phase?

Ans. Requirement Engineering Tasks

- Inception—Establish a basic understanding of the problem and the nature of the solution.
- Elicitation—Draw out the requirements from stakeholders.
- Elaboration—Create an analysis model that represents information, functional, and behavioral aspects of the requirements.
- Negotiation—Agree on a deliverable system that is realistic for developers and customers
- Specification—Describe the requirements formally or informally
- Validation—Review the requirement specification for errors, ambiguities, omissions, and conflicts.
- Requirements management—Manage changing requirements.
- Inception Task • During inception, the requirements engineer asks a set of questions to establish…

5. List the goals of a good design.

Ans. The following are illustrative examples of design goals.

- Usability. Usability **goals** such as a target for the percentage of users who rate a user interface as easy to use.
- Customer Experience. ...
- Brand Image. ...
- Customer Needs. ...
- Customer Perceptions. ...
- Figure of Merit. ...
- Positioning. ...
- Durability.

6. What is scenario-based modeling?

Ans. 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 will refer.

7. What is the purpose of a Data design?

Ans. Database design is defined as a collection of steps that help with designing, creating, implementing, and maintaining a business's data management systems. The main purpose of designing a database is to produce physical and logical models of designs for the proposed database system.
A good database design process is governed by specific rules:

- Distributes your data into tables based on specific subject areas to decrease data redundancy
- Delivers the database the information needed to link the data in the tables
- Provides support, and guarantees the precision and reliability of data
- Caters to your information processing and reporting requirements
- Functions interactively with the database operators as much as possible

8. Differentiate between coupling and cohesion.

Ans.

Cohesion	**Coupling**
Cohesion is the concept of intra module.	Coupling is the concept of inter module.
Cohesion represents the relationship within module.	Coupling represents the relationships between modules.
Increasing in cohesion is good for software.	Increasing in coupling is avoided for software.
Cohesion represents the functional strength of modules.	Coupling represents the independence among modules.
Highly cohesive gives the best	Whereas loosely coupling gives the best

software.	software.
In cohesion, module focuses on the single thing.	In coupling, modules are connected to the other modules.

9. What is software quality Assurance (SQA)?

Ans. Software Quality Assurance (SQA) is simply a way to assure quality in the software. It is the set of activities which ensure processes, procedures as well as standards suitable for the project and implemented correctly. Software Quality Assurance is a process which works parallel to development of a software.

10. Given any 2 differences between Black box & white box testing

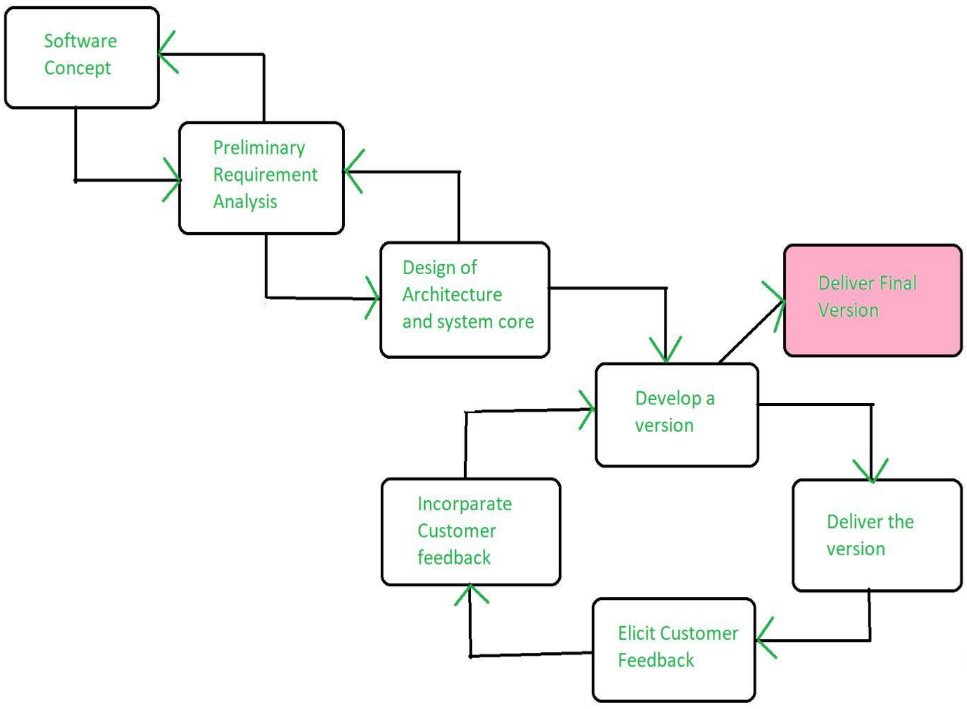
Ans.

Black Box Testing	**White Box Testing**
It is a way of software testing in which the internal structure or the program or the code is hidden, and nothing is known about it.	It is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software.
It is mostly done by software testers.	It is mostly done by software developers.
No knowledge of implementation is needed.	Knowledge of implementation is required.
It can be referred as outer or external software testing.	It is the inner or the internal software testing.
It is functional test of the software.	It is structural test of the software.
This testing can be initiated on the basis of requirement specifications document.	This type of testing of software is started after detail design document.
No knowledge of programming is required.	It is mandatory to have knowledge of programming.
It is the behaviour testing of the software.	It is the logic testing of the software.
It is applicable to the higher levels of testing of software.	It is generally applicable to the lower levels of software testing.
It is also called closed testing.	It is also called as clear box testing.
It is least time consuming.	It is most time consuming.
It is not suitable or preferred for algorithm testing.	It is suitable for algorithm testing.

11. a) Briefly explain evolutionary process models?

b) Explain process framework with an example?

Ans. Evolutionary model is a combination of Iterative and Incremental model of software development life cycle. Delivering your system in a big bang release, delivering it in incremental process over time is the action done in this model. Some initial requirements and architecture envisioning need to be done.


Software Concept
Preliminary Requirement Analysis
Design of Architecture and system core
Develop a version
Deliver Final Version
Deliver the version
Elicit Customer Feedback
Incorparate Customer feedback

- Evolutionary model suggests breaking down of work into smaller chunks, prioritizing them and then delivering those chunks to the customer one by one.
- The number of chunks is huge and is the number of deliveries made to the customer.
- The main advantage is that the customer's confidence increases as he constantly gets quantifiable goods or services from the beginning of the project to verify and validate his requirements.
- The model allows for changing requirements as well as all work in broken down into maintainable work chunks.

Application of Evolutionary Model:

1. It is used in large projects where you can easily find modules for incremental implementation. Evolutionary model is commonly used when the customer wants to start using the core features instead of waiting for the full software.
2. Evolutionary model is also used in object-oriented software development because the system can be easily portioned into units in terms of objects.

Advantages:

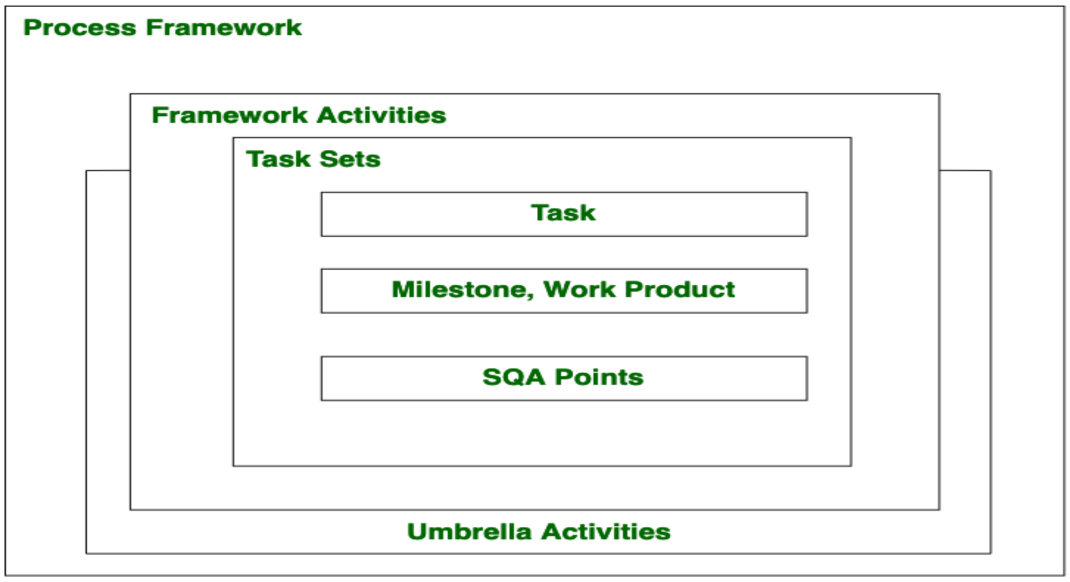
- In evolutionary model, a user gets a chance to experiment partially developed system.
- It reduces the error because the core modules get tested thoroughly.

Disadvantages:

- Sometimes it is hard to divide the problem into several versions that would be acceptable to the customer which can be incrementally implemented and delivered.

b) Explain process framework with an example?

Framework is a Standard way to build and deploy applications. **Software Process Framework** is a foundation of complete software engineering process. Software process framework includes all set of umbrella activities. It also includes number of framework activities that are applicable to all software projects.



Software Process Framework

A generic process framework encompasses five activities which are given below one by one:

1. **Communication:**
   In this activity, heavy communication with customers and other stakeholders, requirement gathering is done.
2. **Planning:**
   In this activity, we discuss the technical related tasks, work schedule, risks, required resources etc.
3. **Modeling:**
   Modeling is about building representations of things in the 'real world'. In modeling activity, a product's model is created in order to better understanding and requirements.
4. **Construction:**
   In software engineering, construction is the application of set of procedures that are needed to assemble the product. In this activity, we generate the code and test the product in order to make better product.
5. **Deployment:**
   In this activity, a complete or non-complete products or software are represented to the customers to evaluate and give feedback. on the basis of their feedback we modify the products for supply better product.

Umbrella activities include:

- Risk management
- Software quality assurance (SQA)
- Software configuration management (SCM)
- Measurement
- Formal technical reviews (FTR)

12. a) Explain about Business process Engineering?

b) What is requirements engineering? Explain about elicitation in detail.

Ans. Business process engineering is a way in which organizations study their current business processes and develop new methods to improve productivity, efficiency, and operational costs. Business process engineering focuses on new business processes, how to diagnose problems with an organization's current methodology, and how to redesign, reconstruct, and monitor processes to ensure they are effective.

Business Process Engineering (BPE) uses a proven systematic approach based on the latest experiences and research to achieve significant improvements. The BPE process helps clients fundamentally rethink and reinvent the business processes needed to achieve the firm's strategic objectives through the maximum use of enabling technologies and organizational strategies. A BPE effort can result in 15% to 50% improvement in performance of the targeted business processes, depending upon whether a reengineering or an improvement approach is used in the effort.

Business Process Engineering Approach
Business Process Engineering (BPE) Approach includes:

- Understanding the Present Mode of Operation (PMO). We'll assemble an experienced team to analyze your current processes, technologies and systems. The result will be creation of a detailed PMO business process model showing interrelationships and dependencies between people, systems, and processes. This will serve as the baseline that proposed changes and actual implementations are evaluated against.
- Determining the Future Mode of Operation (FMO). We'll work with you to build an advisory team to define an FMO business process model based on your business objectives and our combined knowledge of industry best practices.
- Gap Analysis and Transition Plan. A gap analysis of needed business process improvements and transition to the FMO plan will be developed. You'll gain an understanding of the business strategy, timing, personnel, and system/process evolution that will take place.
- Implementation. By trailing and deploying new system or operational process improvements, we'll help you determine whether the intended results will be achieved. If additional system and operational process improvements need to be made, we'll repeat the appropriate PMO, FMO, Gap Analysis and Transition Planning steps as necessary.

B. What is requirements engineering? Explain about elicitation in detail

Ans. The term elicitation is used in books and research to raise the fact that good requirements cannot just be collected from the customer, as would be indicated by the name requirements gathering. Requirements elicitation is non-trivial because you can never be sure you get all requirements from the user and customer by just asking them what the system should do or not do (for Safety and Reliability). Requirements elicitation practices include interviews, questionnaires, user observation, workshops, brainstorming, use cases, role playing and prototyping.

1. '**Problems of scope**'. The boundary of the system is ill-defined, or the customers/users specify unnecessary technical details that may confuse, rather than clarify, overall system objectives.
2. **Problems of understanding**. The customers/users are not completely sure of what is needed, have a poor understanding of the capabilities and limitations of their computing environment, don't have a full understanding of the problem domain, have trouble communicating needs to the system engineer, omit information that is believed to be "**obvious**," specify requirements that conflict with the needs of other customers/users, or specify requirements that are ambiguous or untestable.

3. **Problems of volatility**. The requirements change over time. The rate of change is sometimes referred to as the level of requirement volatility

Requirements quality can be improved through these approaches

1. **Visualization**. Using tools that promote better understanding of the desired end-product such as visualization and simulation.
2. **Consistent language**. Using simple, consistent definitions for requirements described in natural language and use the business terminology that is prevalent in the enterprise.
3. **Guidelines**. Following organizational guidelines that describe the collection techniques and the types of requirements to be collected. These guidelines are then used consistently across projects.
4. **Consistent use of templates**. Producing a consistent set of models and templates to document the requirements.
5. **Documenting dependencies**. Documenting dependencies and interrelationships among requirements.
6. **Analysis of changes**. Performing root cause analysis of changes to requirements and making corrective actions.

13. a) Explain in detail about flow-oriented modeling.

b) Briefly explain software design process and quality.

Ans. A. The flow-oriented modeling represents how data objects are transformed at they move through the system. Derived from structured analysis, flow models use the data flow diagram, a modeling notation that depicts how input is transformed into output as data objects move through the system. Each software function that transforms data is described by a process specification or narrative.

Data Flow Diagram:

The data flow diagram represents the flows of data between different process in a business. It is a graphical technique that depicts information flow and transforms that are applied as data from input to output. It provides a simple, intuitive method for describing business processes without focusing on the details of computer systems. DFDs are attractive techniques because they provide what users do rather than what computers do. In DFD, there are four symbols are used:

1. Process:

The circle represents the process. An activity that changes or transforms data flows. Since they transform incoming data to outgoing data, all processes must have inputs and outputs on a DFD.

2. Data Flow:

The labelled arrows indicate incoming and outgoing data flow. Movement of data between external entities, processes and data stores is represented with an arrow symbol, which indicates the direction of flow.

3. Data Store:

The rectangle represents an external entity. A data store does not generate any operations but simply holds data for later access.

4. External Entity:

In Data Flow Diagrams external entities produce and consume data that flows between the entity and the system being diagrammed.

These data flows are the inputs and outputs of the DFD. Data objects are represented by labeled arrows, and transformations are represented by circles. The DFD is presented in a hierarchical fashion. That is, the first data flow model (sometimes called a level 0 DFD or context diagram) represents the system as a whole. Subsequent data flow diagrams refine the context diagram, providing increasing detail with each subsequent level.

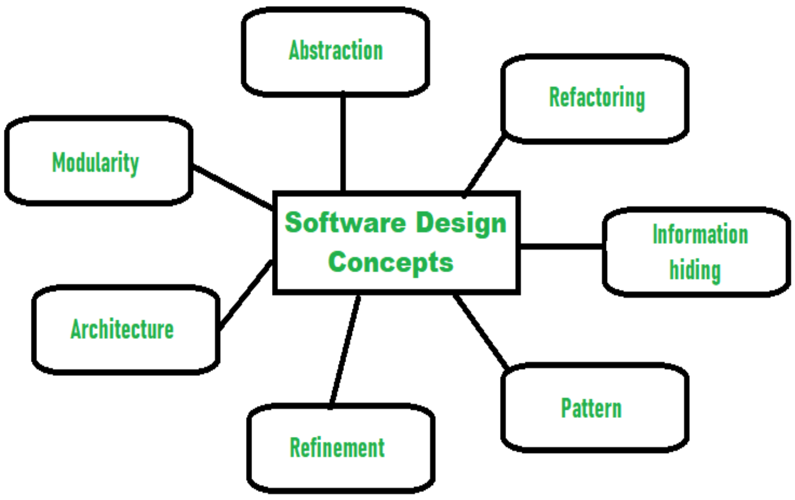
B. Software Design is the process to transform the user requirements into some suitable form, which helps the programmer in software coding and implementation. During the software design phase, the design document is produced, based on the customer requirements as documented in the SRS document. Hence the aim of this phase is to transform the SRS document into the design document. The following items are designed and documented during the design phase:

- Different modules required.
- Control relationships among modules.
- Interface among different modules.
- Data structure among the different modules.
- Algorithms required to implement among the individual modules.

Objectives of Software Design:

1. **Correctness:**
   A good design should be correct i.e. it should correctly implement all the functionalities of the system.
2. **Efficiency:**
   A good software design should address the resources, time and cost optimization issues.
3. **Understandability:**
   A good design should be easily understandable, for which it should be modular and all the modules are arranged in layers.
4. **Completeness:**
   The design should have all the components like data structures, modules, and external interfaces, etc.
5. **Maintainability:**
   A good software design should be easily amenable to change whenever a change request is made from the customer side.

Software Design Concepts:
Concepts are defined as a principal idea or invention that comes in our mind or in thought to understand something. The **software design concept** simply means the idea or principle behind the design. It describes how you plan to solve the problem of designing software, the logic, or thinking behind how you will design software. It allows the software engineer to create the model of the system or software or product that is to be developed or built. The software design concept provides a supporting and essential structure or model for developing the right software. There are many concepts of software design and some of them are given below:



1. **Abstraction- hide Irrelevant data**
   Abstraction simply means to hide the details to reduce complexity and increases efficiency or quality. Different levels of Abstraction are necessary and must be applied at each stage of the design process so that any error that is present can be removed to increase the efficiency of the

software solution and to refine the software solution. The solution should be described in broadways that cover a wide range of different things at a higher level of abstraction and a more detailed description of a solution of software should be given at the lower level of abstraction.

2. **Modularity- subdivide the system**
   Modularity simply means to divide the system or project into smaller parts to reduce the complexity of the system or project. In the same way, modularity in design means to subdivide a system into smaller parts so that these parts can be created independently and then use these parts in different systems to perform different functions. It is necessary to divide the software into components known as modules because nowadays there are different software available like Monolithic software that is hard to grasp for software engineers. So, modularity is design has now become a trend and is also important.
3. **Architecture- design a structure of something**
   Architecture simply means a technique to design a structure of something. Architecture in designing software is a concept that focuses on various elements and the data of the structure. These components interact with each other and use the data of the structure in architecture.
4. **Refinement- removes impurities**
   Refinement simply means to refine something to remove any impurities if present and increase the quality. The refinement concept of software design is actually a process of developing or presenting the software or system in a detailed manner that means to elaborate a system or software. Refinement is very necessary to find out any error if present and then to reduce it.
5. **Pattern- a repeated form**
   The pattern simply means a repeated form or design in which the same shape is repeated several times to form a pattern. The pattern in the design process means the repetition of a solution to a common recurring problem within a certain context.
6. **Information Hiding- hide the information**
   Information hiding simply means to hide the information so that it cannot be accessed by an unwanted party. In software design, information hiding is achieved by designing the modules in a manner that the information gathered or contained in one module is hidden and it can’t be accessed by any other modules.
7. **Refactoring- reconstruct something**
   Refactoring simply means to reconstruct something in such a way that it does not affect the behaviour or any other features. Refactoring in software design means to reconstruct the design to reduce complexity and simplify it without affecting the behaviour or its functions. Fowler has defined refactoring as “the process of changing a software system in a way that it won’t affect the behaviour of the design and improves the internal structure”.

14. Explain Architecture styles and patterns in details

Ans. Architectural Style

The architectural style shows how do we organize our code, or how the system will look like from 10000 feet helicopter view to show the highest level of abstraction of our system design. Furthermore, when building the architectural style of our system we focus on layers and modules and how they are communicating with each other.

There are different types of architectural styles, and moreover, we can mix them and produce a hybrid style that consists of a mix between two and even more architectural styles. Below is a list of architectural styles and examples for each category:

- **Structure architectural styles:** such as layered, pipes and filters and component-based styles.
- **Messaging styles:** such as Implicit invocation, asynchronous messaging and publish-subscribe style.
- **Distributed systems:** such as service-oriented, peer to peer style, object request broker, and cloud computing styles.
- **Shared memory styles:** such as role-based, blackboard, database-centric styles.

- **Adaptive system styles:** such as microkernel style, reflection, domain-specific language styles.

Design Patterns

Design patterns are accumulative best practices and experiences that software professionals used over the years to solve the general problem by – trial and error – they faced during software development. The Gang of Four (GOF, refers to Eric Gamma, Richard Helm, Ralf Johnson, and John Vlissides) wrote a book in 1994 titled with "Design Pattern – Elements of reusable object-oriented software" in which they suggested that design patterns are based on two main principles of object-oriented design:

- Develop to an interface, not to an implementation.
- Favor object composition over inheritance.

Also, they presented that the design patterns set contains 23 patterns and categorized into three main sets:

1. Creational design patterns:
Provide a way to create objects while hiding the creation logic. Thus, the object creation is to be done without instantiating objects directly with the "New" keyword to gives the flexibility to decide which objects need to be created for a given use case. The creational design patterns are:

- **Abstract factory pattern:** provide an interface to create objects without specifying the classes.
- **Singleton pattern:** provide only a single instance of the calls and global access to this instance.
- **Builder Pattern:** Separate the construction from representation and allows the same construction to create multiple representations.
- **Prototype pattern:** creating duplicate without affecting the performance and memory. So, the duplicate object is built from the skeleton of an existing object.

2. Structural patterns:
Concerned with class and object composition. The Structural design patterns are:

- **Adapter pattern:** it works as a bridge between two incompatible interfaces and companies their capabilities.
- **Bridge pattern:** provide a way to decouple the abstraction from its implementation.
- **Filter pattern:** Also known as criteria pattern, it provides a way to filter a set of objects using different criteria and chaining them in a decoupled way through logical operations.
- **Composite pattern:** provide a way to treat a group of objects in a similar way as a single object. It composes objects in term of a tree structure to represent part as well as a whole hierarchy
- **Decorator pattern:** allows adding new functionality to an existing object without altering its structure.
- **Façade pattern:** provide a unified interface to a set of interfaces.it hides the complexities of the system and provides an interface to the client using which the client can access the system.
- **Flyweight pattern:** reduce the number of objects created and to decrease memory footprint and increase performance. It helps in reusing already existing similar kind objects by storing them and creates a new object when no matching object is found.
- **Proxy pattern:** provides a place holder to another object to control access to it. The object has an original object to interface its functionality to the outer world.

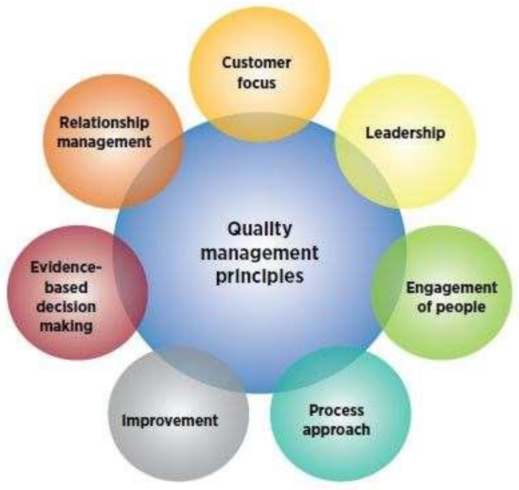
3. Behavioural patterns:
Behavioural patterns are concerned with communications between objects. The following is the list of behavioural patterns:

- **Responsibility pattern:** creates a chain of receiver objects for a request. This pattern decouples the sender and receiver of a request based on the type of request.
- **Command pattern:** it's a data-driven pattern in which A request is wrapped under an object as command and passed to an invoker object.
- **Interpreter pattern:** provides a way to evaluate language grammar or expression. It involves implementing an expression interface that tells to interpret a particular context. This pattern is used in SQL parsing, symbol processing engine, etc.

- **Iterator pattern:** provides a way to access the elements of a collection object in a sequential manner without any need to know its underlying representation.
- **Mediator pattern:** used to reduce communication complexity between multiple objects or classes. It provides a mediator class that normally handles all the communications between different classes and supports easy maintenance of the code by loose coupling.
- **Memento pattern:** used to restore the state of an object to a previous state.
- **Observer pattern:** used when there is a one-to-many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically.
- **State pattern:** is used to change the class behavior based on its state.
- **Null object pattern** helps to avoid null references by having a default object.
- **Strategy pattern:** provides a way to change class behavior or its algorithm at run time.
- **Template pattern:** an abstract class exposes defined way(s)/template(s) to execute its methods. Its subclasses can override the method implementation as per need, but the invocation is to be in the same way as defined by an abstract class.
- **Visitor pattern:** used to change the executing algorithm of an element class.

15. a) Explain ISO 9000 quality standards.

b) What is debugging? Explain the process of debugging.



1. Customer focus
   - Understand the needs of existing and future customers
   - Align organizational objectives with customer needs and expectations
   - Meet customer requirements
   - Measure customer satisfaction
   - Manage customer relationships
   - Aim to exceed customer expectations
   - Learn more about the customer experience and customer satisfaction
2. Leadership
   - Establish a vision and direction for the organization
   - Set challenging goals
   - Model organizational values
   - Establish trust
   - Equip and empower employees
   - Recognize employee contributions
   - Learn more about leadership
3. Engagement of people
   - Ensure that people's abilities are used and valued
   - Make people accountable
   - Enable participation in continual improvement
   - Evaluate individual performance

- Enable learning and knowledge sharing
- Enable open discussion of problems and constraints
- Learn more about employee involvement

4. Process approach
   - Manage activities as processes
   - Measure the capability of activities
   - Identify linkages between activities
   - Prioritize improvement opportunities
   - Deploy resources effectively
   - Learn more about a process view of work and see process analysis tools
5. Improvement
   - Improve organizational performance and capabilities
   - Align improvement activities
   - Empower people to make improvements
   - Measure improvement consistently
   - Celebrate improvements
   - Learn more about approaches to continual improvement
6. Evidence-based decision making
   - Ensure the accessibility of accurate and reliable data
   - Use appropriate methods to analyze data
   - Make decisions based on analysis
   - Balance data analysis with practical experience
   - See tools for decision making
7. Relationship management
   - Identify and select suppliers to manage costs, optimize resources, and create value
   - Establish relationships considering both the short and long term
   - Share expertise, resources, information, and plans with partners
   - Collaborate on improvement and development activities
   - Recognize supplier successes
   - Learn more about supplier quality and see resources related to managing the supply chain

b) What is debugging? Explain the process of debugging.

Ans. Debugging is the process of fixing a bug in the software. In other words, it refers to identifying, analyzing and removing errors. This activity begins after the software fails to execute properly and concludes by solving the problem and successfully testing the software.

Debugging Process: Steps involved in debugging are:

- Problem identification and report preparation.
- Assigning the report to software engineer to the defect to verify that it is genuine.
- Defect Analysis using modelling, documentations, finding and testing candidate flaws, etc.
- Defect Resolution by making required changes to the system.
- Validation of corrections.

Debugging Strategies:

1. Study the system for the larger duration in order to understand the system. It helps debugger to construct different representations of systems to be debugging depends on the need. Study of the system is also done actively to find recent changes made to the software.
2. Backwards analysis of the problem which involves tracing the program backward from the location of failure message in order to identify the region of faulty code. A detailed study of the region is conducting to find the cause of defects.
3. Forward analysis of the program involves tracing the program forwards using breakpoints or print statements at different points in the program and studying the results. The region where the wrong outputs are obtained is the region that needs to be focused to find the defect.
4. Using the past experience of the software debug the software with similar problems in nature. The success of this approach depends on the expertise of the debugger.

16. a) Explain Object Oriented test strategies.

b) Explain metrics for maintenance.

Ans. A. The classical strategy for testing computer software begins with "testing in the small" and works outward toward "testing in the large." Stated in the jargon of software testing, we begin with unit testing, then progress toward integration testing, and culminate with validation and system testing.

Unit Testing in the OO Context
Encapsulation drives the definition of classes and objects. This means that each class and each instance of a class (object) packages attributes (data) and the operations (also known as methods or services) that manipulate these data.

Integration Testing in the OO Context

Because object-oriented software does not have a hierarchical control structure, conventional top-down and bottom-up integration strategies have little meaning. In addition, integrating operations one at a time into a class (the conventional incremental integration approach) is often impossible because of the "direct and indirect interactions of the components that make up the class".

Validation Testing in an OO Context

At the validation or system level, the details of class connections disappear. Like conventional validation, the validation of OO software focuses on user-visible actions and user-recognizable output from the system. To assist in the derivation of validation tests, the tester should draw upon the use-cases that are part of the analysis model. The use-case provides a scenario that has a high likelihood of uncovered errors in user interaction requirements.

Conventional black-box testing methods can be used to drive validations tests. In addition, test cases may be derived from the object-behavior model and from event flow diagram created as part of OOA.

b) Explain metrics for maintenance.

Software metrics can be classified into three categories −

- **Product metrics** − Describes the characteristics of the product such as size, complexity, design features, performance, and quality level.
- **Process metrics** − These characteristics can be used to improve the development and maintenance activities of the software.
- **Project metrics** − This metrics describe the project characteristics and execution. Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity.

Some metrics belong to multiple categories. For example, the in-process quality metrics of a project are both process metrics and project metrics.

Software quality metrics are a subset of software metrics that focus on the quality aspects of the product, process, and project. These are more closely associated with process and product metrics than with project metrics.

Software quality metrics can be further divided into three categories −

- Product quality metrics
- In-process quality metrics
- Maintenance quality metrics

Product Quality Metrics

This metrics include the following −

- Mean Time to Failure
- Defect Density
- Customer Problems
- Customer Satisfaction

Mean Time to Failure

It is the time between failures. This metric is mostly used with safety critical systems such as the airline traffic control systems, avionics, and weapons.

Defect Density

It measures the defects relative to the software size expressed as lines of code or function point, etc. i.e., it measures code quality per unit. This metric is used in many commercial software systems.

Customer Problems

It measures the problems that customers encounter when using the product. It contains the customer's perspective towards the problem space of the software, which includes the non-defect oriented problems together with the defect problems.

18. Explain

(i) Interface Design Steps
2.Validation testing

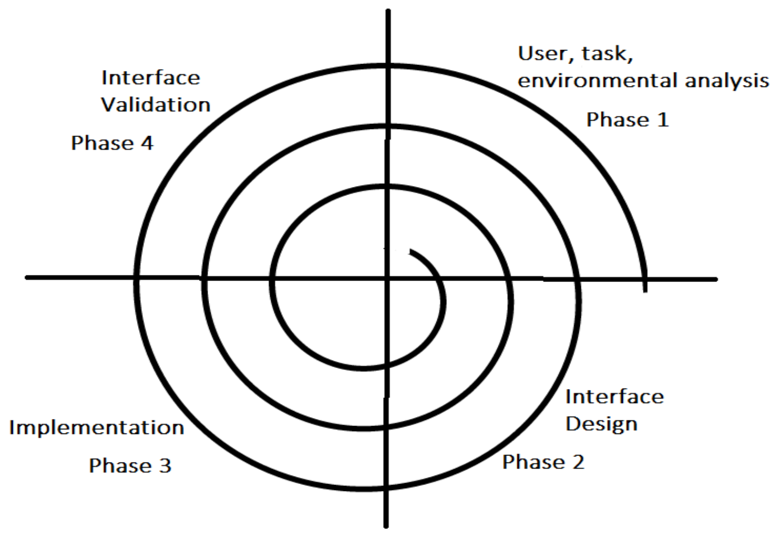
User interface is the front-end application view to which user interacts in order to use the software. The software becomes more popular if its user interface is:

- Attractive
- Simple to use
- Responsive in short time
- Clear to understand
- Consistent on all interface screens

There are two types of User Interface:

1. **Command Line Interface:** Command Line Interface provides a command prompt, where the user types the command and feeds to the system. The user needs to remember the syntax of the command and its use.
2. **Graphical User Interface:** Graphical User Interface provides the simple interactive interface to interact with the system. GUI can be a combination of both hardware and software. Using GUI, user interprets the software.

User Interface Design Process:



The analysis and design process of a user interface is iterative and can be represented by a spiral model. The analysis and design process of user interface consists of four framework activities.

1. **User, task, environmental analysis, and modeling:** Initially, the focus is based on the profile of users who will interact with the system, i.e. understanding, skill and knowledge, type of user, etc, based on the user's profile users are made into categories. From each category requirements are gathered. Based on the requirements developer understand how to develop the interface. Once all the requirements are gathered a detailed analysis is conducted. In the analysis part, the tasks that the user performs to establish the goals of the system are identified, described and elaborated. The analysis of the user environment focuses on the physical work environment. Among the questions to be asked are:
   - Where will the interface be located physically?
   - Will the user be sitting, standing, or performing other tasks unrelated to the interface?
   - Does the interface hardware accommodate space, light, or noise constraints?
   - Are there special human factors considerations driven by environmental factors?
2. **Interface Design:** The goal of this phase is to define the set of interface objects and actions i.e. Control mechanisms that enable the user to perform desired tasks. Indicate how these control mechanisms affect the system. Specify the action sequence of tasks and subtasks, also called a user scenario. Indicate the state of the system when the user performs a particular task. Always follow the three golden rules stated by Theo Mandel. Design issues such as response time, command and action structure, error handling, and help facilities are considered as the design model is refined. This phase serves as the foundation for the implementation phase.
3. **Interface construction and implementation:** The implementation activity begins with the creation of prototype (model) that enables usage scenarios to be evaluated. As iterative design process continues a User Interface toolkit that allows the creation of windows, menus, device interaction, error messages, commands, and many other elements of an interactive environment can be used for completing the construction of an interface.
4. **Interface Validation:** This phase focuses on testing the interface. The interface should be in such a way that it should be able to perform tasks correctly and it should be able to handle a variety of tasks. It should achieve all the user's requirements. It should be easy to use and easy to learn. Users should accept the interface as a useful one in their work.

Golden Rules:
The following are the golden rules stated by Theo Mandel that must be followed during the design of the interface.

Place the user in control:

- Define the interaction modes in such a way that does not force the user into unnecessary or undesired actions: The user should be able to easily enter and exit the mode with little or no effort.
- Provide for flexible interaction: Different people will use different interaction mechanisms, some might use keyboard commands, some might use mouse, some might use touch screen, etc, Hence all interaction mechanisms should be provided.

- Allow user interaction to be interruptable and undoable: When a user is doing a sequence of actions the user must be able to interrupt the sequence to do some other work without losing the work that had been done. The user should also be able to do undo operation.
- Streamline interaction as skill level advances and allow the interaction to be customized: Advanced or highly skilled user should be provided a chance to customize the interface as user wants which allows different interaction mechanisms so that user doesn't feel bored while using the same interaction mechanism.
- Hide technical internals from casual users: The user should not be aware of the internal technical details of the system. He should interact with the interface just to do his work.
- Design for direct interaction with objects that appear on screen: The user should be able to use the objects and manipulate the objects that are present on the screen to perform a necessary task. By this, the user feels easy to control over the screen.

Reduce the user's memory load:

- Reduce demand on short-term memory: When users are involved in some complex tasks the demand on short-term memory is significant. So the interface should be designed in such a way to reduce the remembering of previously done actions, given inputs and results.
- Establish meaningful defaults: Always initial set of defaults should be provided to the average user, if a user needs to add some new features then he should be able to add the required features.
- Define shortcuts that are intuitive: Mnemonics should be used by the user. Mnemonics means the keyboard shortcuts to do some action on the screen.
- The visual layout of the interface should be based on a real-world metaphor: Anything you represent on a screen if it is a metaphor for real-world entity then users would easily understand.
- Disclose information in a progressive fashion: The interface should be organized hierarchically i.e. on the main screen the information about the task, an object or some behavior should be presented first at a high level of abstraction. More detail should be presented after the user indicates interest with a mouse pick.

Make the interface consistent:

- Allow the user to put the current task into a meaningful context: Many interfaces have dozens of screens. So it is important to provide indicators consistently so that the user know about the doing work. The user should also know from which page has navigated to the current page and from the current page where can navigate.
- Maintain consistency across a family of applications: The development of some set of applications all should follow and implement the same design, rules so that consistency is maintained among applications.
- If past interactive models have created user expectations do not make changes unless there is a compelling reason.

2.Validation testing

Software Testing - Validation Testing

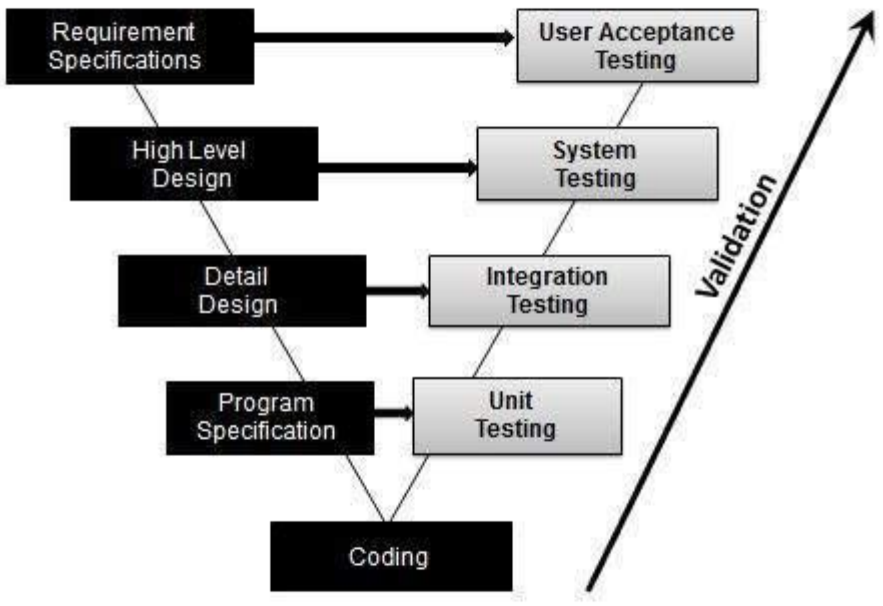
The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements.

Validation Testing ensures that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment.

It answers to the question, Are we building the right product?

Validation Testing - Workflow:

Validation testing can be best demonstrated using V-Model. The Software/product under test is evaluated during this type of testing.



Activities:

- Unit Testing
- Integration Testing
- System Testing
- User Acceptance Testing