MODULE 2

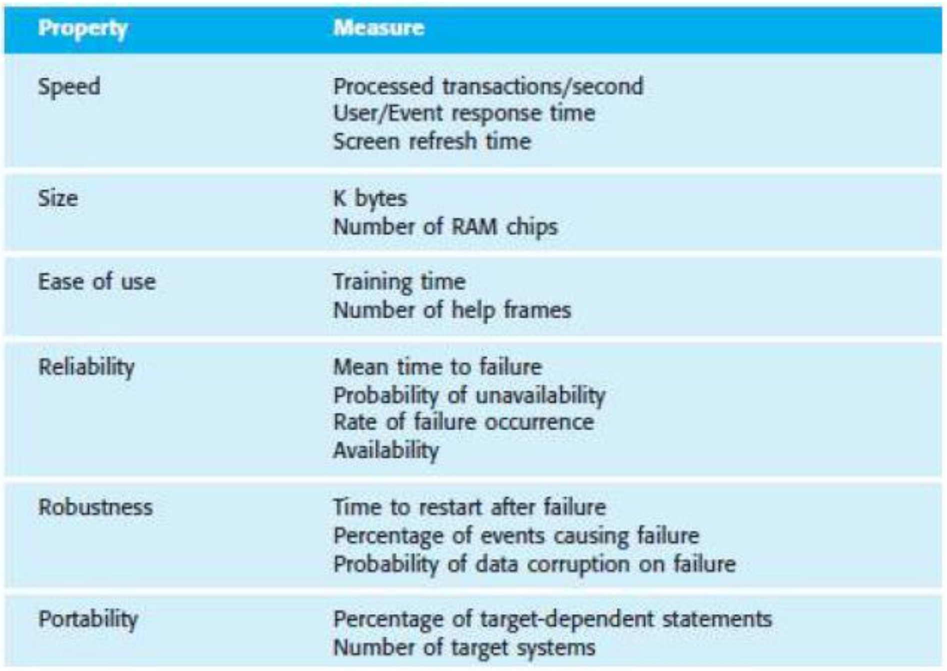
REQUIREMENT ANALYSIS AND DESIGN

IMPORTANT QUESTIONS

1. Difference between functional and non functional requirements.

Functional Requirements	Non Functional Requirements
A functional requirement defines a system or its component.	A non-functional requirement defines the quality attribute of a software system.
It specifies "What should the software system do?"	It places constraints on "How should the software system fulfill the functional requirements?". Timing constraints,constraints on the development process,constrints imposed by standards.
Functional requirement is specified by User.	Non-functional requirement is specified by technical peoples e.g. Architect, Technical leaders and software developers.

2. Explain the metrics for specifying functional requirements?



Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	K bytes Number of RAM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target-dependent statements Number of target systems

3. Why requirements elicitation is considered as a critical task in requirements engineering?

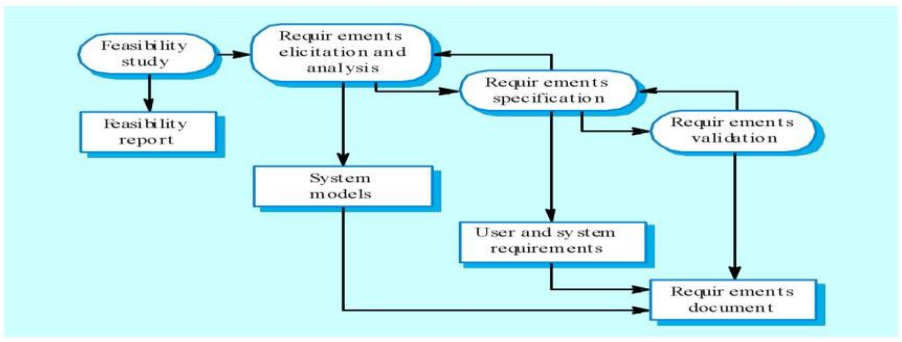
Requirement Elicitation is a very difficult task. The process of requirements elicitation is generally accepted as one of the critical activities in the Requirement Engineering process. Requirement Elicitation is process and normally considered as a process of finding out what are the real need of the customer from the system. Getting the right requirements is considered as a vital but difficult part of software development projects. Requirement Elicitation is important and fundamental aspect in Software development. Many problems occur at development and maintenance is due to poor requirement gathering, management and requirement change management. Certain techniques are used for requirement elicitation.

4. **List the elements of requirement engineering process. ?**

Requirement Engineering Process It is a four step process, which includes –

Feasibility Study, Requirement Gathering, Software Requirement Specification, Software Requirement Validation.

The requirements engineering process



©Ian Sommerville 2006 **Software Engineering, 8th edition. Chapter 7** Slide 5

- The goal is to create and maintain a system requirement document.
- The overall process includes 4 sub-processes:
  - **1) Feasibility Study:** Concern with accessing whether system is useful to the business.
  - **2) Elicitation & Analysis:** Discovering requirement and analyzing them.
  - **3) Specialization:** Converting this requirement into some standard form.
  - **4) Validation:** Checking that the requirement actually defines the system that the customer wants.
- These activities are concerned with
  - → discovery documentation &
  - → checking of requirements.
- The people involved in development gain better understanding of
  - → what they want to do the software
  - → modification made to the software and organizational environment.
- The process of managing these changing requirements is called **requirement management**.

5. Explain any three methods for requirements elicitation.

Requirements discovery (elicitation techniques)
The process of gathering information about the required and existing systems and distilling the user and system requirements from this information. Interaction is with system stakeholders from managers to external regulators. Systems normally have a range of stakeholders.

1. INTERVIEWs
2. SCENARIOS
3. ETHNOGRAPHY

1. **Formal or informal interviews with stakeholders are part of most RE processes. Types of interview**

 Closed interviews : stakeholders answers based on pre-determined list of questions
 Open interviews : in which there is no predefined agenda, where various issues are explored with stakeholders

Effective interviewing
 Be open-minded, avoid pre-conceived ideas about the requirements and are willing to listen to stakeholders.

 Prompt the interviewee to get discussions going using a springboard question, a requirements proposal, or by working together on a prototype system.

2.Scenarios are real-life examples of how a system can be used. They should include
 A description of the starting situation;
 A description of the normal flow of events;
 A description of what can go wrong;
 Information about other concurrent activities;
 A description of the state when the scenario finishes**.**

3.Ethnography
A social scientist spends a considerable time observing and analyzing how people actually work. People do not have to explain or articulate their work.
 Social and organizational factors of importance may be observed.
 Ethnographic studies have shown that work is usually richer and more complex than suggested by simple system models.
 Requirements that are derived from cooperation and awareness of other people's activities.
 Awareness of what other people are doing leads to changes in the ways in which we do things.
 Ethnography is effective for understanding existing processes but cannot identify new features that should be added to a system.

6. What are the contents we should contain in the SRS document and design document

SRS Document SRS document is a contract between the development team and the customer. Once the SRS document is approved by the customer, any subsequent controversies are settled by referring the SRS document. SRS document defines the customer's requirements in terms of Functions, performance, external interfaces and design constraints.

SRS Includes: • Functional • Non functional • User • Interface • System Design Document

The purpose of a design is to describe how the enhancements will be incorporated into the existing project. It should contain samples of the finished product. This could include navigational mechanism screenshots, example reports, and component diagrams.

Design Includes: • E-R Diagrams • Data flow diagrams • Data Dictionary.

7. Identify any four type of requirement defined for a software system?.

User Requirements – User requirements are natural language statements that may be accompanied by diagrams that show both the services the system is expected to provide the users, as well as any constraints the system will operate under.

System Requirements – System requirements are more detailed descriptions of the software system's functions, services, and operational constraints. The exact implementation should be defined and this document can serve as part of a contract between the system buyer and the developers

Functional Requirements – Functional requirements are the statements of the services that the system should provide, how the system handles specific input and how the system should behave in a given situation.

Nonfunctional Requirements – Nonfunctional requirements refer to the contraints on the system's services or functions. These can include industry standards that must be followed such as timing, security, etc. An example being that bank transactions most likely have to have encrypted communications for certain operations, if not all. These constraints usually effect the entire system rather than certain features or services.

8. List the components of SRS.

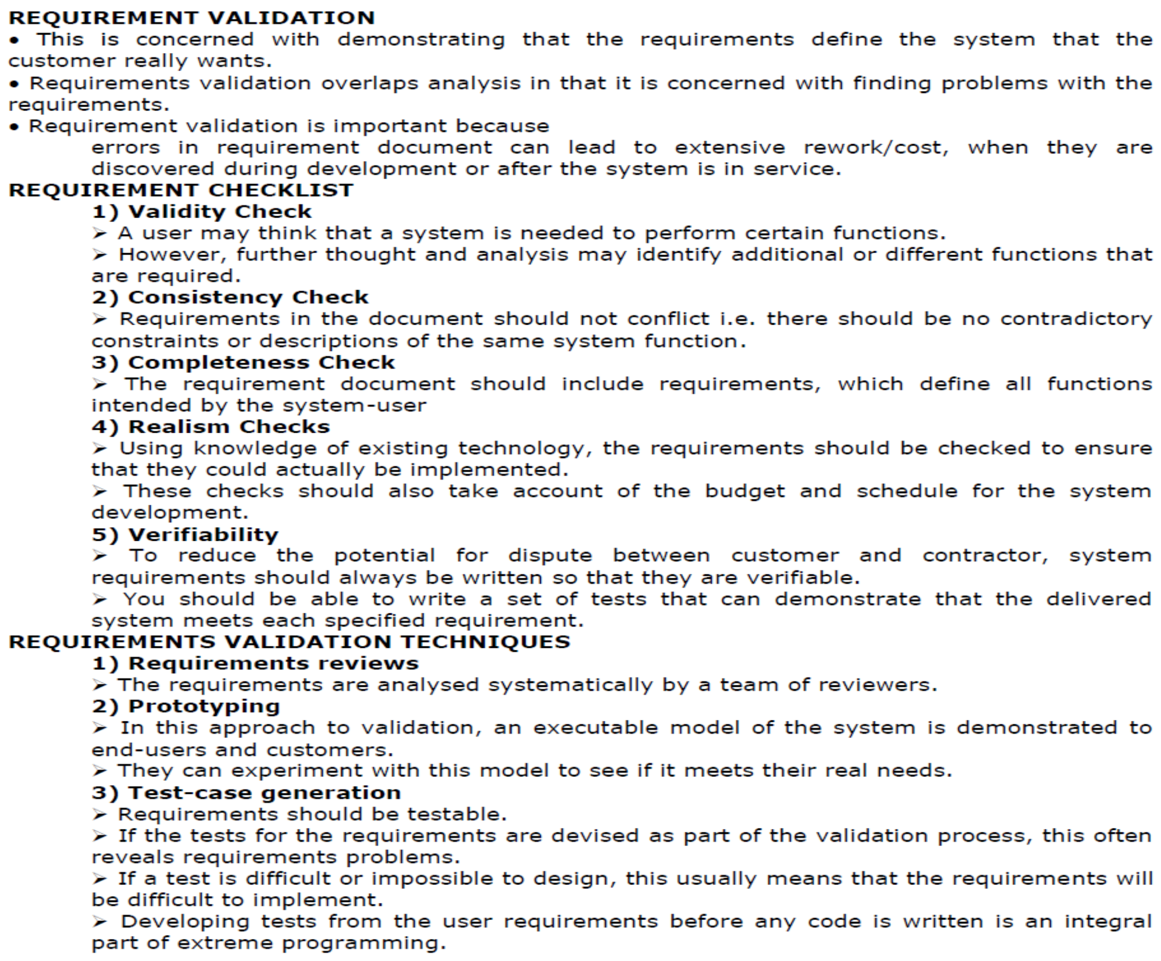
Refer SRS template (for correct order)

3c. Explain the structure of the requirement document. (08 Marks)
Ans:
STRUCTURE OF A REQUIREMENTS DOCUMENT
1) Preface
• This should
→ define the expected readership of the document &
→ describe document's version history & a summary of changes made in each version.
2) Introduction
• This should
→ describe the need for the system.
→ describe system's functions and explain how it will work with other systems.
→ describe how the system fits into the overall business objectives of the organisation.
3) Glossary
• This should define the technical terms used in the document.
4) User Requirements
• This should
→ describe services provided for the user
→ describe the non-functional definition system requirements &
→ specify product standards which must be followed.
• This description may use
→ natural language
→ diagrams or
→ other notations.
5) System Architecture
• This should
→ describe a high-level overview of the anticipated system-architecture
→ show the distribution of functions across system modules &
→ highlight architectural components that are reused.
6) System Requirements
• This should describe the functional and non-functional specification requirements in more detail.
7) System models
• This should select one or more system models showing the relationships between
→ system-components and
→ system.
• These might be
→ object models
→ data-flow models and
→ semantic data models.
8) System Evolution
• This should describe anticipated changes due to hardware evolution, changing user needs, etc.
9) Appendices
• These should provide detailed, specific information which is related to the application.
• Examples: i) Hardware description and ii) Database description.
i) Hardware Requirements define the minimal & optimal configurations for the system.
ii) Database Requirements define the logical organisation of the data used by the system and the relationships between data.
10) Index
• Several indexes to the document may be included.
• Along with a normal alphabetic index, there may be an index of diagrams, an index of functions, etc.

9. Write the role of users of requirement document

ROLES OF USERS OF REQUIREMENT DOCUMENT
1) System Customers
• Specify the requirements and read them to check that they meet their needs.
• Customers specify changes to the requirements.
2) Managers
• Use the requirements document to
→ plan a bid for the system and
→ plan the system development process
3) System Engineers
• Use the requirements to understand what system is to be developed.
4) System Test Engineers
• Use the requirements to develop validation tests for the system.
5) System Maintenance Engineers
• Use the requirements to understand the system and the relationships between its parts.

10. Explain requirement validation?



REQUIREMENT VALIDATION

• This is concerned with demonstrating that the requirements define the system that the customer really wants.

• Requirements validation overlaps analysis in that it is concerned with finding problems with the requirements.

• Requirement validation is important because

errors in requirement document can lead to extensive rework/cost, when they are discovered during development or after the system is in service.

REQUIREMENT CHECKLIST

1) Validity Check

➢ A user may think that a system is needed to perform certain functions.

➢ However, further thought and analysis may identify additional or different functions that are required.

2) Consistency Check

➢ Requirements in the document should not conflict i.e. there should be no contradictory constraints or descriptions of the same system function.

3) Completeness Check

➢ The requirement document should include requirements, which define all functions intended by the system-user

4) Realism Checks

➢ Using knowledge of existing technology, the requirements should be checked to ensure that they could actually be implemented.

➢ These checks should also take account of the budget and schedule for the system development.

5) Verifiability

➢ To reduce the potential for dispute between customer and contractor, system requirements should always be written so that they are verifiable.

➢ You should be able to write a set of tests that can demonstrate that the delivered system meets each specified requirement.

REQUIREMENTS VALIDATION TECHNIQUES

1) Requirements reviews

➢ The requirements are analysed systematically by a team of reviewers.

2) Prototyping

➢ In this approach to validation, an executable model of the system is demonstrated to end-users and customers.

➢ They can experiment with this model to see if it meets their real needs.

3) Test-case generation

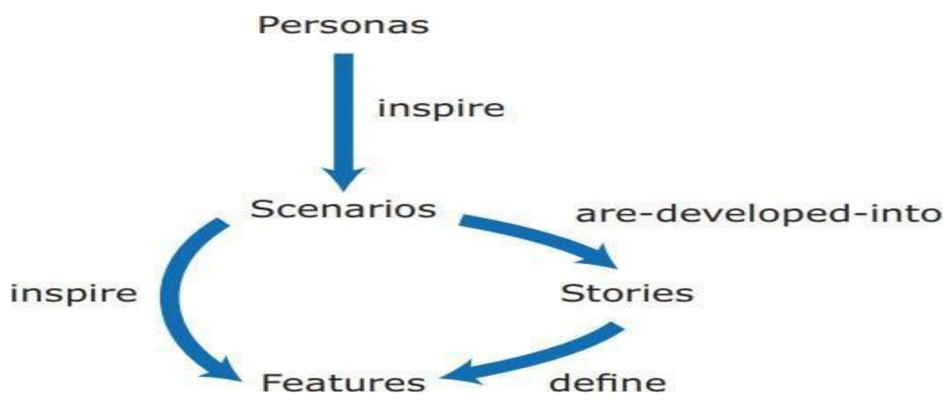
➢ Requirements should be testable.

➢ If the tests for the requirements are devised as part of the validation process, this often reveals requirements problems.

➢ If a test is difficult or impossible to design, this usually means that the requirements will be difficult to implement.

➢ Developing tests from the user requirements before any code is written is an integral part of extreme programming.

11. Explain about Personas, use cases and scenarios ? (refer notes)



Personas are about "imagined users," character portraits of types of user that you think might adopt your product.

 Ex: if your product is aimed at managing appointments for dentists, you might create a dentist persona, a receptionist persona, and a patient persona.

SCENARIOS

 A scenario is a narration that describes a situation in which a user is using your product's features to do something that they want to do.

 Scenarios are used in the design of requirements and system features, in system testing, and in user interface design.

User Stories

These are finer-grain narratives that set out in a more detailed and structured way a single thing that a user wants from a software system.

User stories are not intended for planning but for helping with feature identification.

Feature Identification

A feature is a way of allowing users to access and use your product's functionality so that the feature list defines the overall functionality ofthe system.

Feature is a fragment of functionality that implements some user or system need. We can access features through user interface of a product.

Feature is something that the user needs or wants.

12. Define software architecture?.

The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.

Identify three key reasons that software architecture is important:

Software architecture provides a representation that facilitates communication among all stakeholders.

The architecture highlights early design decisions that will have a profound impact on all software engineering work that follows.

Architecture "constitutes a relatively small, intellectually graspable model of how the system is structured and how its components work together"

13. Define software component?

A component is a modular building block for computer software. The OMG Unified Modeling Language Specification defines a component as "a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces."

14. List out the architectural styles? (Refer Notes- Important)

- Data-Centered Architecture
- Data-Flow Architectures
- Call and Return Architectures:
- Object-Oriented Architectures
- Layered Architectures

15. Define tracebility matrix.

A traceability matrix is a document that details the technical requirements for a given test scenario and its current state. It helps the testing team understand the level of testing that is done for a given product.

The traceability process itself is used to review the test cases that were defined for any requirement. It helps users identify which requirements produced the most number of defects during a testing cycle.

16. Discuss the steps for architectural design ? (Refer Notes for Explanation- important)

1. Representing the System in Context
2. Defining Archetypes
3. Refining the Architecture into Components
4. Describing Instantiations of the System.

17. Difference between Cohesion and coupling

Cohesion is an indication of the relative functional strength of a module. A cohesive module performs a single task, requiring little interaction with other components in other parts of a program.

Coupling is an indication of the relative interdependence among modules. Coupling depends on the interface complexity between modules, the point at which entry or reference is made to a module, and what data pass across the interface.

18. Explain the Types of cohesion and coupling ?

Coincidental cohesion

- A module is said to have coincidental cohesion if it performs a set of tasks that relate to each other very loosely, if at all.
- In this case, the module contains a random collection of functions. It is likely that the functions have been put in the module out of pure coincidence without any thought or design.
- For example, in a transaction processing system (TPS), the get-input, print-error, and summarize- members
- Functions are grouped into one module.

Logical cohesion

- A module is said to be logically cohesive, if all elements of the module perform similar operations, e.g. error handling, data input, data output, etc.
- An example of logical cohesion is the case where a set of print functions generating different output reports are arranged into a single module.

Temporal cohesion

- When a module contains functions that are related by the fact that all the functions must be executed in the same time span, the module is said to exhibit temporal cohesion.
- The set of functions responsible for initialization, start-up, a shutdown of some process, etc. exhibit temporal cohesion.

Procedural cohesion

- A module is said to possess procedural cohesion, if the set of functions of the module are all part of a procedure (algorithm) in which a certain sequence of steps have to be carried out for achieving an objective, e.g. the algorithm for decoding a message.

Communicational cohesion

- A module is said to have communicational cohesion, if all functions of the module refer to or update the same data structure, e.g. the set of functions defined on an array or a stack.

Sequential cohesion

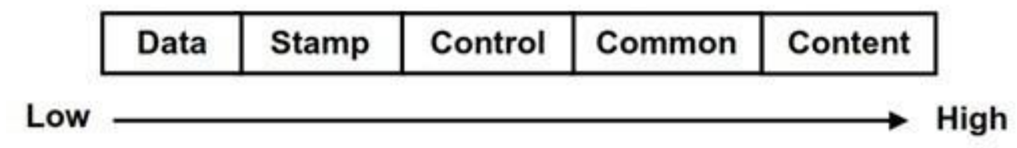
- A module is said to possess sequential cohesion if the elements of a module form the parts of the sequence, where the output from one element of the sequence is input to the next.
- For example, in a TPS, the get-input, validate-input, sort-input functions are grouped into one module.

Functional cohesion

- Functional cohesion is said to exist if different elements of a module cooperate to achieve a single function. For example, a module containing all the functions required to manage employees' pay-roll exhibits functional cohesion.
- Suppose a module exhibits functional cohesion and we are asked to describe what the module does, then we would be able to describe it using a single sentence.

Coupling

Classification of Coupling



Data coupling

- Two modules are data coupled if they communicate through a parameter. An example is an elementary data item passed as a parameter between two modules, e.g. an integer, float, character, etc.
- This data item should be problem-related and not used for the control purpose.

Stamp coupling

- Two modules are stamp coupled if they communicate using a composite data item such as a record in PASCAL or a structure in C.

Control coupling

- Control coupling exists between two modules if data from one module is used to direct the order of instructions executed in another.
- An example of control coupling is a flag set in one module and tested in another module.

Common coupling

- Two modules are commonly coupled if they share data through some global data items. Content coupling
- Content coupling exists between two modules if they share code, e.g. a branch from one module into another module.

17. Define Use case ? .Differentiate between Primary actor and secondary actor.?

Use Case: A use case is a methodology used in system analysis to identify, clarify and organize system requirements. The use case is made up of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal.

The primary actor is the one that initiates a use case and a secondary actor is the one that helps completion of the use case through his specific support.

18. Explain design concepts of software Engineering ?(Refer notes for examples)

- Software design sits at the technical core of software engineering and is applied regardless of the software process model that is used.
- The design task produces a data design, an architectural design, an interface design, and a component design.

Abstraction

- A solution is stated in large terms using the language of the problem environment at the highest level abstraction.
- The lower level of abstraction provides a more detail description of the solution.
- A sequence of instruction that contain a specific and limited function refers in a procedural abstraction.
- A collection of data that describes a data object is a data abstraction.

Architecture

- The complete structure of the software is known as software architecture.
- Structure provides conceptual integrity for a system in a number of ways.
- The architecture is the structure of program modules where they interact with each other in a specialized way.
- The components use the structure of data.
- The aim of the software design is to obtain an architectural framework of a system.
- The more detailed design activities are conducted from the framework.

Patterns

- A design pattern describes a design structure and that structure solves a particular design problem in a specified content.

Modularity

- A software is separately divided into name and addressable components. Sometime they are called as modules which integrate to satisfy the problem requirements.
- Modularity is the single attribute of a software that permits a program to be managed easily.

Information hiding

- Modules must be specified and designed so that the information like algorithm and data presented in a module is not accessible for other modules not requiring that information.

Functional independence

- The functional independence is the concept of separation and related to the concept of modularity, abstraction and information hiding.
- The functional independence is accessed using two criteria i.e Cohesion and coupling.
- Cohesion: Cohesion is an extension of the information hiding concept.
- A cohesive module performs a single task and it requires a small interaction with the other components in other parts of the program.
- Coupling: Coupling is an indication of interconnection between modules in a structure of software.

Refinement

- Refinement is a top-down design approach.
- It is a process of elaboration.
- A program is established for refining levels of procedural details.

- A hierarchy is established by decomposing a statement of function in a stepwise manner till the programming language statement are reached.

Refactoring

- It is a reorganization technique which simplifies the design of components without changing its function behavior.
- Refactoring is the process of changing the software system in a way that it does not change the external behavior of the code still improves its internal structure.

Design classes

- The model of software is defined as a set of design classes.
- Every class describes the elements of problem domain and that focus on features of the problem which are user visible.

19. List and state the basic design principles

The Open-Closed Principle (OCP). "A module [component] should be open for extension but closed for modification.

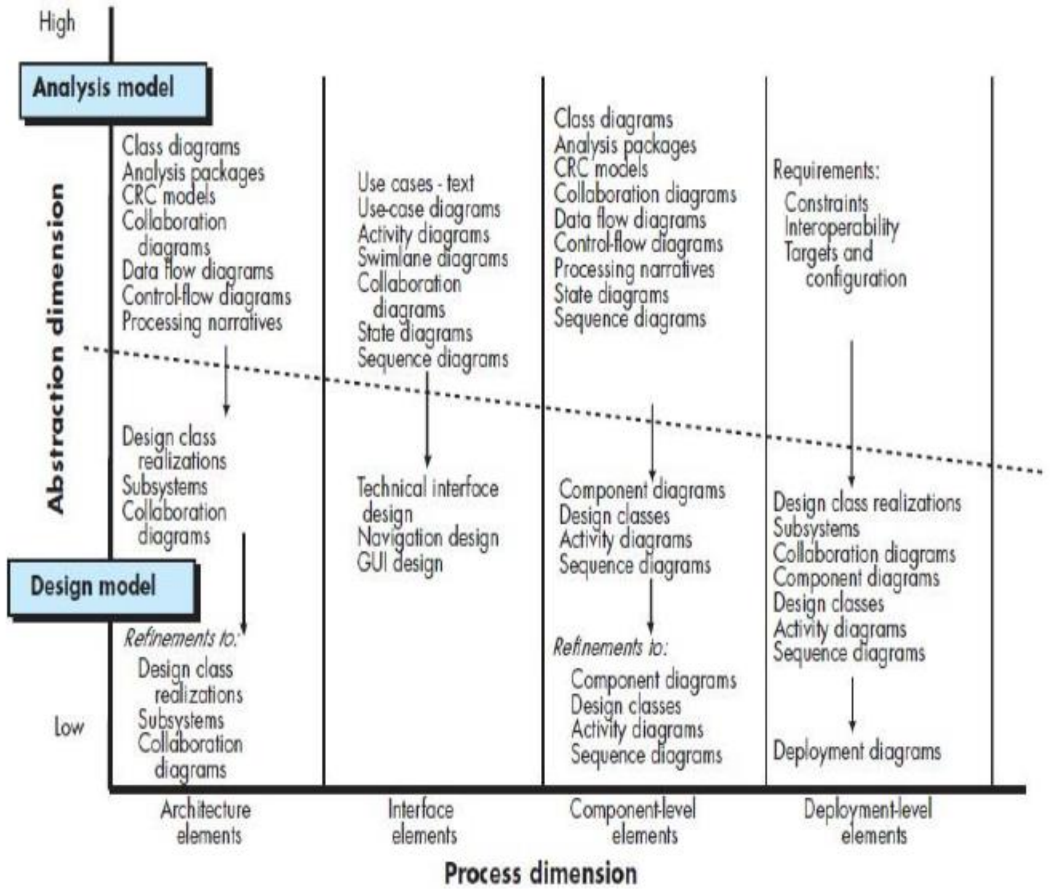
The Liskov Substitution Principle (LSP). "Subclasses should be substitutable for their base classes".

The Interface Segregation Principle (ISP). "Many client-specific interfaces are better than one general purpose· interface"

The Release Reuse Equivalency Principle (REP). "The granule of reuse is the granule of release"

20 . Explain Design Model (refer notes for figures)

The design model can be viewed in two different dimensions.



The process dimension indicates the evolution of the design model as design tasks are executed as part of the software process.

The abstraction dimension represents the level of detail as each element of the analysis model is transformed into a design equivalent and then refined iteratively. The dashed line indicates the boundary between the analysis and design models. The design model can be viewed in two different dimensions. The process dimension indicates the evolution of the design model as design tasks are executed as part of the software process. The abstraction dimension represents the level of detail as each element of the analysis model is transformed into a design equivalent and then refined iteratively. The dashed line indicates the boundary between the analysis and design models.

Data Design Elements

Data design (sometimes referred to as data architecting) creates a model of data and/or information that is represented at a high level of abstraction (the customer/user's view of data). This data model is then refined into progressively more implementation-specific representations that can be processed by the computer-based system.

Architectural Design Elements

The architectural design for software is the equivalent to the floor plan of a house. The floor plan depicts the overall layout of the rooms; their size, shape, and relationship to one another; and the doors and windows that allow movement into and out of the rooms. The floor plan gives us an overall view of the house. Architectural design elements give us an overall view of the software

Interface design elements.(interface diagram)

The interface design elements for software depict information flows into and out of a system and how it is communicated among the components defined as part of the architecture.

Component-Level Design Elements (draw component diagram)

The component-level design for software is the equivalent to a set of detailed drawings (and specifications) for each room in a house. These drawings depict wiring and plumbing within each room, the location of electrical receptacles and wall switches, faucets, sinks, showers, tubs, drains, cabinets, and closets, and every other detail associated with a room

Deployment-Level Design Elements.

Deployment-level design elements indicate how software functionality and subsystems will be allocated within the physical computing environment that will support the software.

21. List the steps to conduct component level design

The following steps represent a typical task set for component-level design, when it is applied for an object-oriented system.
Step1. Identify all design classes that correspond to the problem domain. Using the requirements and architectural model, each analysis class and architectural component is elaborated
Step 2. Identify all design classes that correspond to the infrastructure domain. These classes are not described in the requirements model and are often missing from the architecture model, but they must be described at this point. Classes and components in this category include GUI components (often available as reusable components), operating system components, and object and data management components.
Step3. Elaborate all design classes that are not acquired as reusable components. Elaboration requires that all interfaces, attributes, and operations necessary to implement the class be described in detail. Design heuristics (e.g., component cohesion and coupling) must be considered as this task is conducted.
Step3a. Specify message details when classes or components collaborate. The requirements model makes use of a collaboration diagram to show how analysis classes collaborate with one another. Messages that are passed between objects within a system.
Step3b. Identify appropriate interfaces for each component. Within the context of component-level design, a UML interface is "a group of externally visible (i.e., public) operations. The interface contains no internal structure, it has no attributes, no associations. ".
Step3c. Elaborate attributes and define data types and data structures required to implement them. In general, data structures and types used to define attributes are defined within the context of the programming language that is to be used for implementation.
Step3d. Describe processing flow within each operation in detail. This may be accomplished using a programming language-based pseudo code or with a UML activity diagram. Each software component is elaborated through a number of iterations that apply the stepwise refinement concept.

The first iteration defines each operation as part of the design class. In every case, the operation should be characterized in a way that ensures high cohesion; that is, the operation should perform a single targeted function or sub function. The next iteration does little more than expand the operation name.

Step 4. Describe persistent data sources (databases and files) and identify the classes required to manage them. Databases and files normally transcend the design description of an individual component. In most cases, these persistent data stores are initially specified as part of architectural design. However, as design elaboration proceeds, it is often useful to provide additional detail about the structure and organization of these persistent data sources.

Step 5. Develop and elaborate behavioural representations for a class or component. UML state diagrams were used as part of the requirements model to represent the externally observable behaviour of the system and the more localized behaviour of individual analysis classes. During component-level design, it is sometimes necessary to model the behaviour of a design class.

Step6. Elaborate deployment diagrams to provide additional implementation detail. Deployment diagrams are used as part of architectural design and are represented in descriptor form. In this form, major system functions are represented within the context of the computing environment that will house them. During component-level design, deployment diagrams can be elaborated to represent the location of key packages of components. However, components generally are not represented individually within a component diagram. In some cases, deployment diagrams are elaborated into instance form at this time. This means that the specific hard- ware and operating system environment(s) that will be used is (are) specified and the location of component packages within this environment is indicated.

Step7. Refactor every component-level design representation and always con- sider alternatives. Design is an iterative process. The first component-level model we create will not be as complete, consistent, or accurate as the nth iteration you apply to the model. It is essential to refactor as design work is conducted.

22. List out the architectural considerations of software design

Economy —many software architectures suffer from unnecessary complexity driven by the inclusion of unnecessary features or nonfunctional requirements (e.g., reusability when it serves no purpose). The best software is uncluttered and relies on abstraction to reduce unnecessary detail.

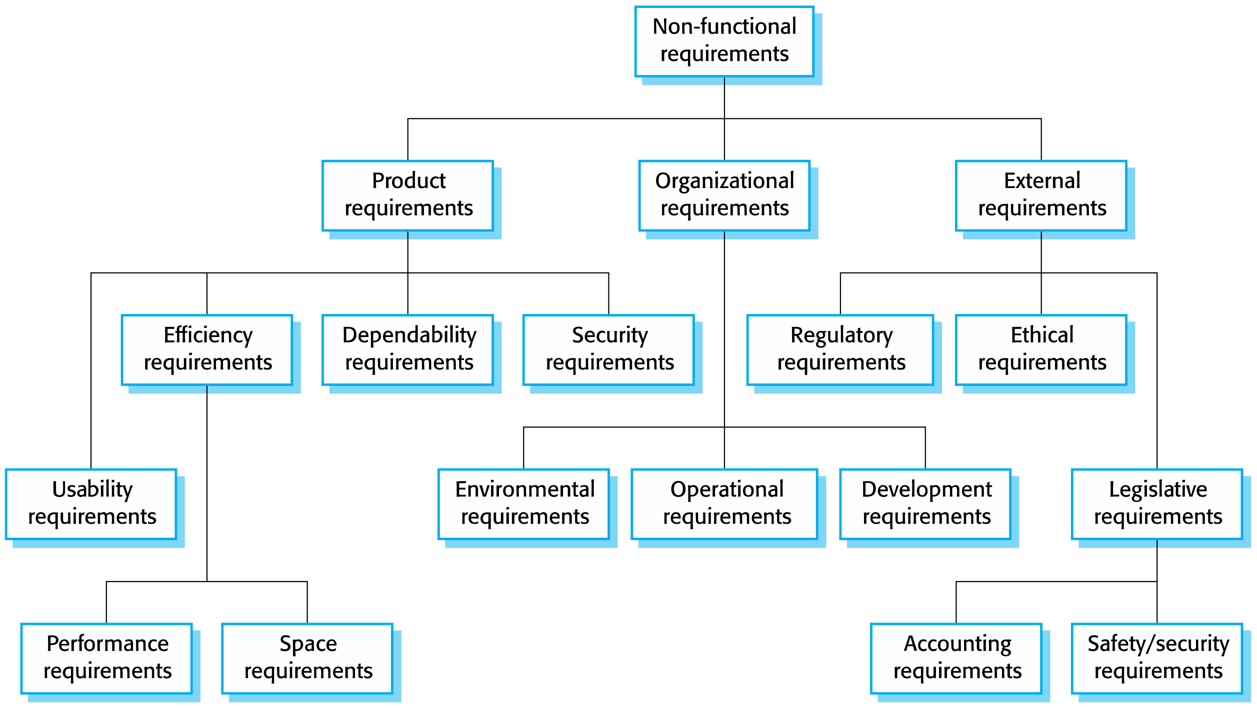
Visibility —As the design model is created, architectural decisions and the reasons for¬ them should be obvious to software engineers who examine the model at a later time. Poor visibility arises when important design and domain concepts are poorly communicated to those who must complete the design and implement the system.

Spacing— Separation of concerns in a design without introducing hidden dependencies is¬ a desirable design concept that is sometimes referred to as spacing. Sufficient spacing leads to modular designs, but too much spacing leads to fragmentation and loss of visibility.

Symmetry —Architectural symmetry implies that a system is consistent and balanced in¬ its attributes. Symmetric designs are easier to understand, comprehend, and communicate. As an example of architectural symmetry, consider a customer account object whose life cycle is modeled directly by a software architecture that requires both open () and close() methods. Architectural symmetry can be both structural and behavioral. Emergence —Emergent, self-organized behavior and control are often the key to creating¬ scalable, efficient, and economic software architectures. For example, many real-time software applications are event driven. The sequence and duration of the events that define the system's behavior is an emergent quality. It is very difficult to plan for every possible sequence of events. Instead the system architect should create a flexible system that accommodates this emergent behavior.

22. List the different types of functional requirements?

- ✧ Product requirements
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
- ✧ Organisational requirements
  - Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.
- ✧ External requirements
  - Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.



3c. What is Ethnography? How ethonography is effective in discovering the types of requirement? (08 Marks)

Ans:

ETHNOGRAPHY

• Ethnography is an observational technique that can be used to understand social and organisational requirements (Figure 7.9).

• An analyst immerses himself in the working environment where the system will be used.

• The analyst

→ observes the day-to-day work and

→ notes down actual tasks in which participants are involved.

• Ethnography helps analysts discover implicit system requirements that reflect the actual processes in which people are involved.

How Ethnography Benefits

1) People often find it very difficult to articulate details of their work because it is second nature to them.

2) People understand their own work but may not understand its relationship with other work in the organisation.

3) Social and organisational factors that affect the work but that are not obvious to individuals may only become clear when noticed by an unbiased observer.

• Two types of requirements:

1) Requirements that are derived from the way in which people actually work

For example:

➢ Air traffic controllers may switch off an aircraft conflict alert system that detects aircraft with intersecting flight paths.

➢ Their control strategy is designed to ensure that these aircraft are moved apart before problems occur.

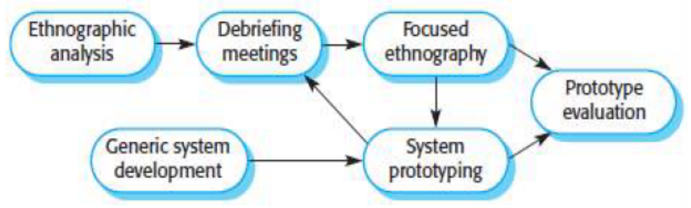
2) Requirements that are derived from cooperation and awareness of other people's activities.

For example:

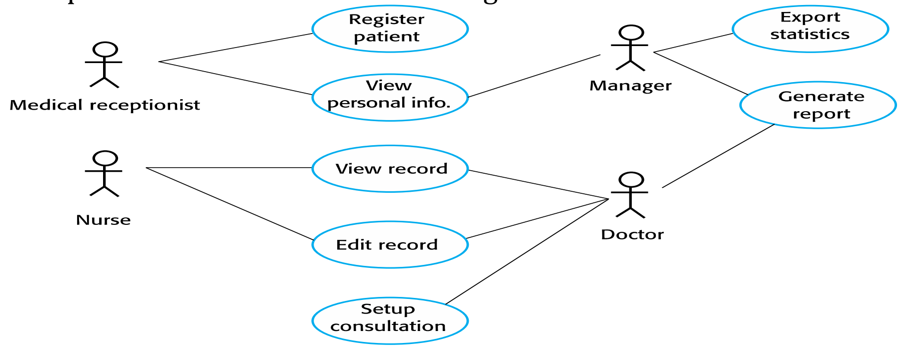
➢ Air traffic controllers may use an awareness of other controllers' work to predict the number of aircraft that will be entering their control sector.

➢ They then modify their control strategies depending on that predicted workload.

➢ Therefore, an automated ATC system should allow controllers in a sector to have some visibility of the work in adjacent sectors.



23. explain Use case with a use case diagram



24. Explain about the functional and nonfunctional requirements of library management system.

1. Any library member should be able to search books by their title, author, subject category as well by the publication date.
2. Each book will have a unique identification number and other details including a rack number which will help to physically locate the book.
3. There could be more than one copy of a book, and library members should be able to check-out and reserve any copy. We will call each copy of a book, a book item.
4. The system should be able to retrieve information like who took a particular book or what are the books checked-out by a specific library member.
5. There should be a maximum limit (5) on how many books a member can check-out.
6. There should be a maximum limit (10) on how many days a member can keep a book.
7. The system should be able to collect fines for books returned after the due date.
8. Members should be able to reserve books that are not currently available.
9. The system should be able to send notifications whenever the reserved books become available, as well as when the book is not returned within the due date.
10. Each book and member card will have a unique barcode. The system will be able to read barcodes from books and members' library cards.

Non functional requirements

1. Error handling : LMS product shall handle expected and non-expected errors in ways that prevent loss in information and long downtime period.

2. Performance Requirements. The system shall accommodate high number of books and users without any fault.• Responses to view information shall take no longer than 5 seconds to appear on the screen.

3. Safety Requirements: System use shall not cause any harm to human users.

4. Security Requirements: System will use secured database

• Normal users can just read information but they cannot edit or modify anything except their• personal and some other information. System will have different types of users and every user has access constraints.•