DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

YEAR / SEMESTER: II / IV
SUB NAME: SOFTWARE ENGINEERING **SUB CODE: CS6403**

PART A

1. What is software engineering?
Software engineering is a discipline in which theories, methods and tools are applied to develop professional software.

2. What is Software?
Software is nothing but a collection of computer programs that are related documents that are indented to provide desired features, functionalities and better performance.

3. What are the characteristics of the software?
- Software is engineered, not manufactured.
- Software does not wear out.
- Most software is custom built rather than being assembled from components.

4.What is requirement engineering?
Requirement engineering is the process of establishing the services that the customer requires from the system and the constraints under which it operates and is developed.

5. Define the computer based system.
The computer based system can be defined as "a set or an arrangement of elements that are organized to accomplish some predefined goal by processing information".

6. What does Verification represent?
Verification represents the set of activities that are carried out to confirm that the software correctly implements the specific functionality.

7. What does Validation represent?
Validation represents the set of activities that ensure that the software that has been built is satisfying the customer requirements.

8. What are the merits of incremental model?
i. The incremental model can be adopted when tere are less number of people involved in the project.
ii. Technical risks can be managed with each increment.
iii. For a very small time span, at least core product can be delivered to the customer.

9. What are the fundamental activities of a software process?
- Specification
- Design and implementation
- Validation
- Evolution

10. What are the challenges in software?
- Copying with legacy systems.
- Heterogeneity challenge
- Delivery times challenge

Part - B

1. Explain Water fall Model.

The first published model of the software development process was derived from other engineering process. This is illustrated from following figure. Because of the cascade from one phase to another, this model is known as "Water fall model" or "Software life cycle model". The principle stage of the model map on to fundamental development activities.

Requirement analysis and definition:

The systems services constraints and goals are established by consultation with system users. They are then defined in detail and serve as system specification

System and Software Design:

The system design process partition the requirements to either hardware or software systems. It establishes overall system architecture. Software design involves identifying and describing the fundamental software system abstraction and their relationship.

Implementation and unit testing:

During this stage the software design is realized as a set of programs or program unit. Unit testing involves verifying that each unit meets its specification

Integration and system testing:

The individual program units or programs are integrated and tested as a complete system to ensure that the software requirement has been met.

2. Explain the Concept of software process.

The following are the basic principles to develop a software process.

(i) Structured set of activities required e.g. Specification, Design, Validation, Evolution Activities vary depending on the organization and the type of system being developed (iii)It Must be explicitly modelled if it is to be managed

In order to analysis software process we must clearly mention the Process Characteristics

i)Understandability
(ii) Visibility
(iii)Supportability
(iv) Acceptability
(v) Reliability
(vi) Maintainability

Generic – building is a type of software building that build any product some points to remember are:

(i)Specification - set out requirements and constraints
(ii) Design - Produce a paper model of the system

3. Explain about prototyping

Model. Prototyping is used in 2

Types they are

Evolutionary prototyping: Objective is to work with customers and to evolve a final system from an initial outline specification. Should start with well-understood requirements

Throw-away prototyping: Objective is to understand the system requirements. Starts with poorly understood requirements are difficult to capture accurately.

Problems with Prototyping

(i) Documentation may get neglected
(ii) Effort in building a prototype may be wasted (iii)Difficult to plan and manage

Advantages

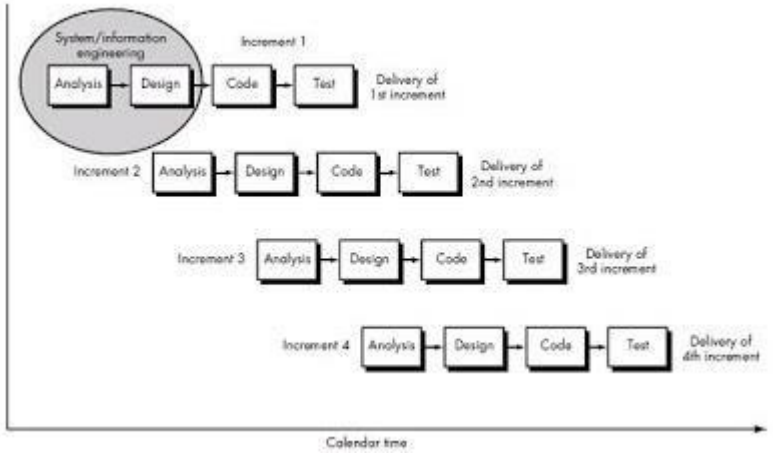
(i) Faster than the waterfall model
(ii) High level of user involvement from start
(iii) Technical or other problems discovered early - risk reduced
Strategies (iv)Evolutionary Prototyping

Problems with Evolutionary Prototyping

(i) Difficult to plan as amount of effort is uncertain
(ii) Documentation may be neglected
(iii) Could degenerate into ―Build and Fix‖ with poorly structured code
(iv)Languages which are good for prototyping not always best for final product

Advantages

(i)Effort of prototype is not wasted
(ii) faster than the waterfall model
(iii) High level of user involvement from start
(iv) Technical or other problems discovered early - risk reduced



4. Explain in detail about different Software Process model.

Incremental model. Have same phases as the waterfall model.
Phases are Analysis.
Design. Code. Test.
Incremental model delivers series of releases to customers called as increments.
The first increment is called as core product. Here only the document processing facilities are available. Second increment, more sophisticated document producing and processing facilities are available.
Next increment spelling and grammar checking facilities are given.

Merits

This model can be adopted when there is less number of people involved in the project. Technical risks can be managed with each increment.
For a very small time span, at least core product can be delivered to the customer.

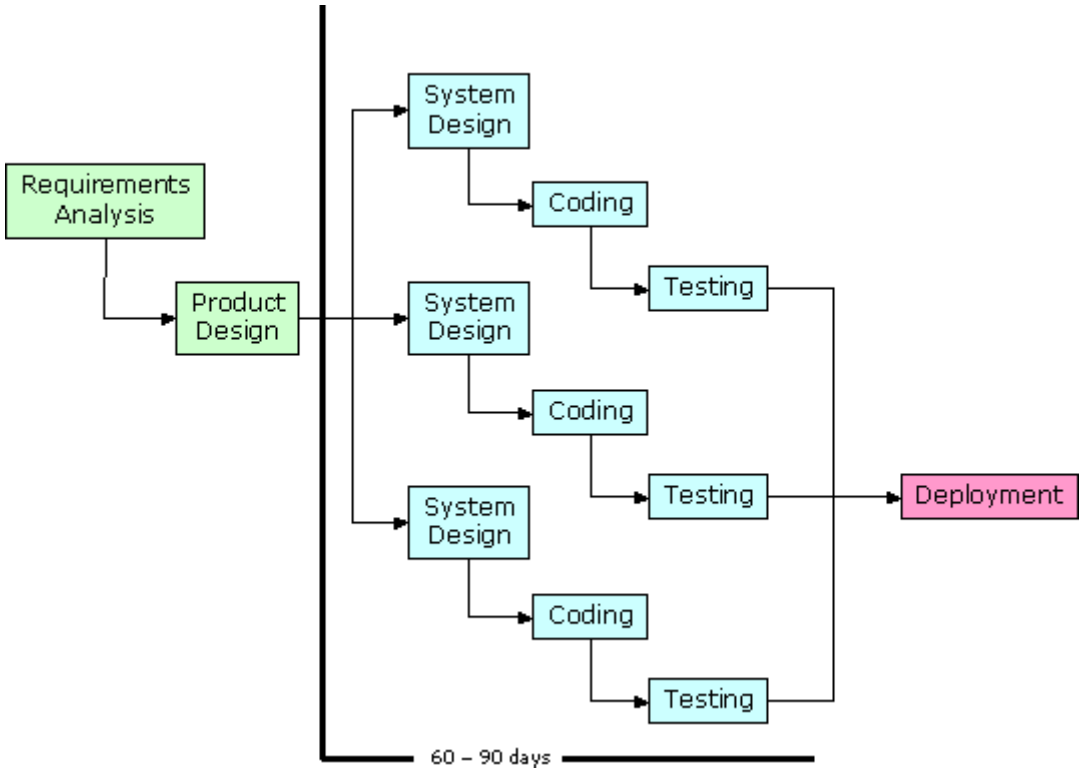
RAD Model

Rapid Application Development Model is the type of incremental model. Construction.

Phases

Business
modeling Data
modeling
Process
modeling

Application generation. Testing and turnover.



Water fall Model. The first published model of the software development process was derived from other engineering process. This is illustrated from following figure. Because of the cascade from one phase to another, this model is known as "Water fall model" or "Software life cycle model". The principle stage of the model map on to fundamental development activities.

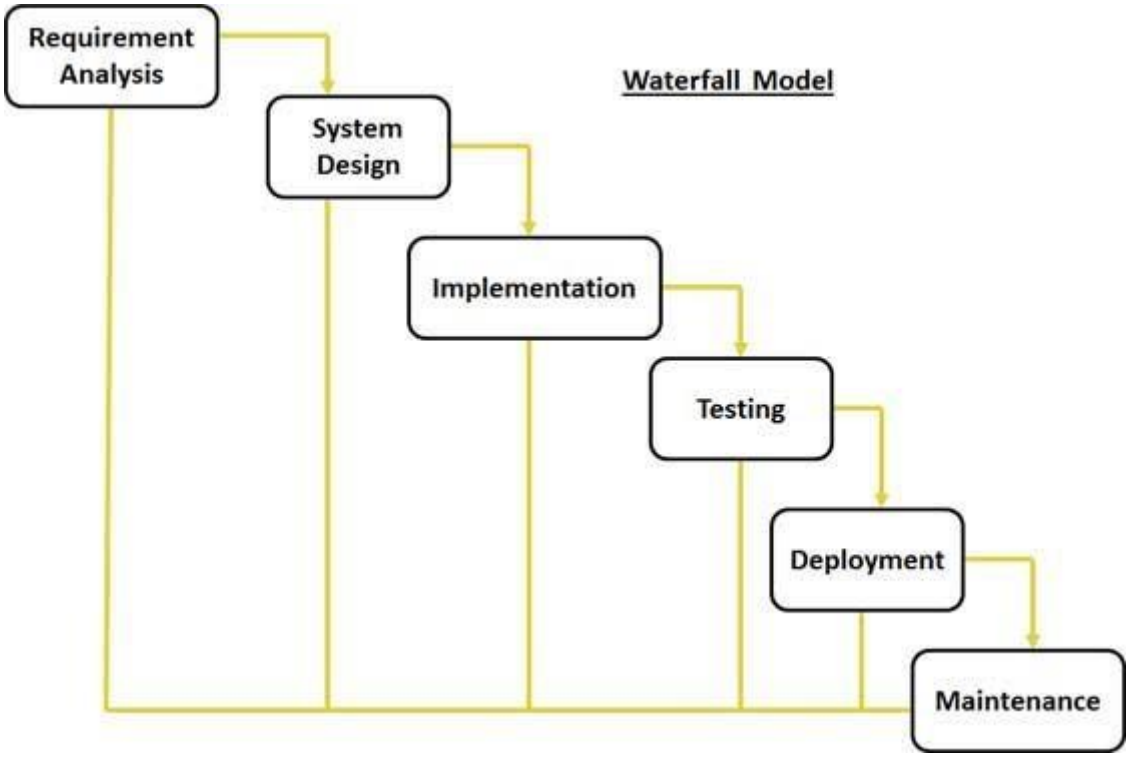
(i) Requirement analysis and definition: The systems services constraints and goals are established by consultation with system users. They are then defined in detail and serve as system specification.

(ii) System and Software Design: The system design process partition the requirements to either hardware or software systems. It establishes an overall system architecture. Software design involves identifying and describing the fundamental software system abstraction and their relationship.

(iii) Implementation and unit testing: During this stage the software design is realized as a set of programs or program unit. Unit testing involves verifying that each unit meets its specification

(iv) Integration and system testing: The individual program units or programs are integrated and tested as a complete system to ensure that the software requirement has been met. After testing the software system is delivered to the customer.

(v) Operation and maintenance: The system is installed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages of life cycle.



Waterfall Model

Iterative waterfall model

- The iterative waterfall model is as shown in the following figure. Requirement gathering phase in which all requirements are identified. The design phase is responsible for creating architectural view of the software.
- The implementation phase in which the software design is transformed into coding.
- Testing is a kind of phase in which the developed software component is fully tested.
- Maintenance is an activity by which the software product can be maintained. Requirements Design implementation Testing Maintenance

SPIRAL MODEL

- The spiral model is divided into number of frame works. These frameworks are denoted by task regions.
- Usually there are six task regions. In spiral model project entry point axis is defined.
- The task regions are: Customer communication Planning
- Risk analysis. Engineering.
- Construct and release.
- System engineering process follows a waterfall model for the Customer evaluation.

Drawbacks

It is based on customer communication. It demands considerable risk assessment**.**

5. Explain in detail about the life cycle process.

System requirements definition

Three types of requirements Abstract functional requirements. System properties. Undesirable Characteristics. System objectives System requirement problem.

The system design process Process steps

Partition requirements Identify sub-systems.
Assign requirements to sub-systems. Specify sub-system functionality. Define sub-system interfaces. Requirement
Definition System Design
Sub-system Design System Integration
System Decommissioning
System evolution System Installation

Sub-System development process
After system design it starts.
Involve use of COTS (Commercial-Off-The-Shelf).
System Integration
It is the process of putting hardware, software and people together to make a system.

6. Identify the umbrella activities in software engineering process.

The umbrella activities in a software development life cycle process include the following:

1. **Software Project Management**
2. **Formal Technical Reviews**
3. **Software Quality Assurance**
4. **Software Configuration Management**
5. **Re-usability Management**
6. **Risk Management**
7. **Measurement and Metrics**
8. **Document Preparation and Production**

Framework Activities
An effective process model should define a small set of framework activities that are always applicable, regardless of project type. The APM defines the following set of framework activities:

• **project definition** - tasks required to establish effective communication between developer and customer(s) and to define requirements for the work to be performed

• **planning** - tasks required to define resources, timelines and other project related information and assess both technical and management risks

• **engineering and construction** - tasks required to create one or more representations of the software (can include the development of executable models, i.e., prototypes or simulations) and to generate code and conduct thorough testing

• **release** - tasks required to install the software in its target environment, and provide customer support (e.g., documentation and training)

• **customer use** - tasks required to obtain customer feedback based on use and evaluation of the deliverables produced during the release activity
Each of the above framework activities will occur for every project. However, the set of tasks (we call this a *task set*) that is defined for each framework activity will vary depending upon the project type (e.g., Concept Development Projects will have a different task set than Application Enhancement Projects) and the degree of rigor selected for the project.

8.Software Project Management:

Software Project Management

• Software project management is especially difficult

• Software project management : The Manager's

View Process/Project/Product/People

People Project

RFP Process

Product Methods

Tools Metrics

• Numerical measures that quantify the degree to which software, a process or a project possesses a given Attribute

• Metrics help the followings

– Determining software quality level

– Estimating project schedules

– Tracking schedule process

– Determining software size and complexity

– Determining project cost

– Process improvement Software Metrics

• Without measure it is impossible to make a plan, detect problems, and improve a process and product

• A software engineer collects measure and develops metrics so that indicators will be obtained

• An indicator provides insight that enables the project manager or software engineers to adjust the process, the project, or the product to make things better

Software Metrics

• The five essential, fundamental metrics:

– Size (LOC, etc.)

– Cost (in dollars)

– Duration (in months)

– Effort (in person-month)

– Quality (number of faults

detected) Product Size Metrics

• Conventional metrics

– Size-oriented metrics

– Function-oriented metrics

– Empirical estimation models

• Object-Oriented metrics

– Number of scenario scripts

– Number of key classes

– Number of support classes

– Average number of support classes per key classes

• User-Case oriented

metrics Product Size

Metrics (cont'd)

• Web engineering product metrics

– Number of static web pages

– Number of dynamic web pages

– Number of internal page links

– Number of persistent page

links Estimate Uncertainty

UNIT II REQUIREMENTS ANALYSIS AND SPECIFICATION

PART A

1. Define software prototyping.
Software prototyping is defined as a rapid software development for validating the requirements

2. What are the benefits of prototyping?
i. Prototype serves as a basis for deriving system specification.
ii. Design quality can be improved.
iii. System can be maintained easily.
iv. Development efforts may get reduced.
v. System usability can be improved.

3. What are the prototyping approaches in software process?
i. **Evolutionary prototyping –** In this approach of system development, the initial prototype is prepared and it is then refined through number of stages to final stage.

ii. **Throw-away prototyping –** Using this approach a rough practical implementation of the system is produced. The requirement problems can be identified from this implementation. It is then discarded. System is then developed using some different engineering paradigm.

4. What are the advantages of evolutionary prototyping?
i. Fast delivery of the working system.
ii. User is involved while developing the system.
iii. More useful system can be delivered.
iv. Specification, design and implementation work in coordinated manner.

5. What are the various Rapid prototyping techniques?
i. Dynamic high level language development.
ii. Database programming.
iii. Component and application assembly.

6. What is the use of User Interface prototyping?
This prototyping is used to pre-specify the look and feel of user interface in an effective way.

7. What is a data object?
Data object is a collection of attributes that act as an aspect, characteristic, quality, or descriptor of the object.

8. What are attributes?
Attributes are the one, which defines the properties of data object.

9. What is cardinality in data modeling?
Cardinality in data modeling, cardinality specifies how the number of occurrences of one object is related to the number of occurrences of another object.

10. Define Data Dictionary.
The data dictionary can be defined as an organized collection of all the data elements of the system with precise and rigorous definitions so that user and system analyst will have a common understanding of inputs, outputs, components of stores and intermediate calculations.

PART B

1. Explain Requirement Elicitation.

In its simplest possible form the requirements elicitation process might be considered as a conversation between the client and the software engineer that results in an understanding by the software engineer of what the customer wants. Life is seldom that simple however and experience has shown that although eliciting requirement is easiest when there is a good understanding of the application domain. It is essential that the requirements engineer understands the application domain but how do you know that you understand it? You might think that when the user says a crank hard and support the Fox-1 unless you know what they mean but how can you be sure?

One way is to gain experience in the application
Domain (more about this in the lecture) but there are problems with that. The knowledge Clients have is built up over years do you have years to spend on requirement elicitation? Also the client's knowledge may be spread amongst many people so you would inevitably need to gain experiences in many different aspects of the clients business. So experience of the user's domain, whilst clearly valuable and useful, cannot be the only answer. There is simply not the time for the requirements engineer to become expert in the application domain. Though the return for investment on some familiarization is usually large.

The answer is to develop an explicit conceptual model of the domain and to present that back to the client for verification

Requirements Elicitation might be described as eliciting a specification of what is required by allowing experts in the problem domain to describe the goals to be reached during the problem resolution. This being so, we may be limited to having a vague desire at the beginning of the process, such as "We want a new aircraft carrier", and at the end of the process having a detailed description of the goals and some clear idea of the steps necessary to reach the goals

There are several things wrong with this description. Where does the logical model reside, in people's heads? Is there an expert with sufficient breadth and depth of domain knowledge to ensure the goal and all its sub goals are consistent and achievable? If there is not, are we merely leaving to the design stage the process of systematizing the sub goals. It is very likely that many goals will be inconsistent, even deliberately contradictory. Can we say the Requirements Elicitation stage is complete while this is so? We certainly can if our methods for supporting the elicitation have no means of establishing the consistency of the goals. How precise do we need to be in specifying our goals.

2.Explain about Requirement Validation.

Requirement validation ensures captured requirements reflect the functionality desired by the customer and other stakeholders. Although requirement validation is not the focus of requirement testability analysis, it is supported. Requirement validation involves an engineer, user or customer judging the validity (i.e. correctness) of each requirement. Models provide a means for stakeholders to precisely understand the requirements and assist in recognizing omissions. Test automatically derived from the model support requirement validation through manual inspection or execution within simulation or host environments.
Requirement validation is the final stage of requirements engineering.

Purpose of Requirement Validation

The aim of requirement validation is to validate the requirements, i.e., check the requirements to certify that they represent an acceptable description of the system, which is to be implemented. Distinction Between Requirement Analysis and Requirement Validation Requirement analysis is concerned with requirement as elicited from system stakeholders. The requirements are usually incomplete and are expressed in an informal and unstructured way. Requirement validation is concerned with checking a final draft of a requirements document which includes all system requirements and where known incompleteness and inconsistency has been removed. Different Concerns Between Requirement Analysis

and Requirement Validation Requirements analysis should mostly be concerned with answering the question have we got the right requirement

Requirements validation is mostly be concerned with answering the question Requirements Validation Process The main problem of requirements validation is that there is no exiting document, which can be a basis for the validation. A design or a program may be validated against the specification. However, there is no way to demonstrate that a requirements specification is correct with respect to some other system representation. Specification validation, therefore, really means ensuring that the requirements document represent a clear description of the system for design and implementation and is a final check that the requirements meet stakeholder needs.

3.Explain about Requirement Management.

Requirements management is the process of managing changes to a system‘s requirements. Normally there is more than 50% of a system‘s requirements which will be modified before it is put into service. The Principal Concerns of Requirements Management
The principal concerns of requirements management are:

- To manage changes to agreed requirements
- To manage the relationship between requirements, and To manage the dependencies between the requirements document and other documents produced during the
- Systems and software engineering process. Factors Leading to

Requirements change The following factors are main causes leading to requirement changes:

Requirements errors, conflicts and inconsistencies evolving customers/end-user knowledge of the system Technical, schedule or cost problems changing customer priorities Environmental changes Organizational changes. Stable and Volatile Requirements. Requirements changes occur while the requirements are being elicited, analyzed and validated and after the system has gone into service Although change is inevitable, it is usually the case that some requirements are more stable than others. Stable requirements are concerned with the essence of a system and its application domain. They change more slowly than volatile requirements.

Volatile requirements are specific to the instantiation of the system in a particular environment and for a particular customer. Volatile requirements are easily changed. Generally, there are four types of volatile requirements:

Mutable requirements - which change because of changes to the environment in which the system is operating. Emergent requirements - which cannot be completely defined when the system is specified but which emerge as the system is designed and implemented. Consequential requirements - which are based on assumptions about how the system will be used. When the system is put to use, some of these assumptions will be wrong. Users need to adapt to the system and find new ways to use its functionality. Compatibility requirements - which depend on other equipment or processes. As this equipment changes, these requirements also evolve. Requirement Identification and Storage

4.Write a short notes on Data Dictionary

Data Dictionary is also called data repository Documents specific facts about the system .

The Following are the important divisions of data flows:

(i)Data flows
(ii)Data stores
(iii)Processes
(iv)External entities
(v) Data structures (records)
(vi) Data elements (data items, fields)
(vii) (vii)Documenting the elements

All of the items in a DFD is to be documented in the Data Dictionary, also called the Data Repository. All major characteristics of the items must be recorded and described. The key objective is to provide clear, Comprehensive information about the system.

During the documentation process, paper-based standard forms or a CASE tool can be used. Various tools are available, and Visible Analyst is a popular example.

Documenting the data elements

Must document every data element. Example attributes of a data element are:

(i)Name or label
(ii) Alternate name(s)
(iii) Type and length
(iv)(iv)Output format
(v) (v)Default value
(vi)Prompt header or field caption
(vii)Source
(viii)Security
(ix)Responsible user(s)
(x)Acceptable values and data validation
(xi) Derivation formula
(xii)Description and comments

Documenting the data flows
Must document every data flow. Example attributes of a data flow are:

(i)Name or label
(ii)Alternate name(s)
(iii)Description
(iv)Origin
(v)Record – group of data elements
(vi)Volume and frequency

Documenting the external entities
Must document every external entity. Example attributes of a external entity are:
(i)Name or label
(ii)Alternate name(s)
(iii)Description
(iv)Input data flows
(v)Output data flows

Documenting the records
Must document every record. Example attributes of a record are:
(i)Name or label
(ii)Alternate name(s)
(iii)Definition or description
(iv)Record content or composition
(v)Data Dictionary Reports
Must document every process. Example attributes of a process are:
Data dictionary serves as a central storehouse for documentation Using this data, you can produce many valuable reports through data dictionary.

Name	Alias	Use	Content Description	Supplement
Telephone number	None	Phone setup dial phone	Telephone number= local number long distance number] local number= prefix+ access number long distance number= 0+ area code + local number prefix = [2520/2524] access number any three digit string area code =[141,151,]	None

5.Explain in detail about software requirement document?

Software requirement document Software requirement specification 1.introduction
-purpose of this document
-scope of this document
-Overview 2.General description
3. Functional requirements
-Description
-criticality
-technical issues
-cost and schedule
- risks
-Dependencies with other requirements
-any other appropriate 4.Interface requirements
-user interface
-GUI
-CLI
-API
-hardware interface
-communication interface
-Software communication 5.performance requirements 6.design constraints
7.other non-functional attrib 8.operational scenarios 9.preliminary schedule 10.preliminary budget 11.appendices

- Characteristics of srs
- Example of SRS

6.Requirements document users

System customers

Specify the requirements and read them back to check that they meet their needs; specify changes to the requirements

Development Managers

Use the requirements document to plan a bid for the system and to plan the

system development process

Implementation Programmers Use the requirements to understand what system is to be developed

Test programmers

Use the requirements to develop validation tests for the system

Maintenance programmers

Use the requirements to help understand the system and the relationships between its parts

7. Explain in detail about Non-functional requirements

Product requirements

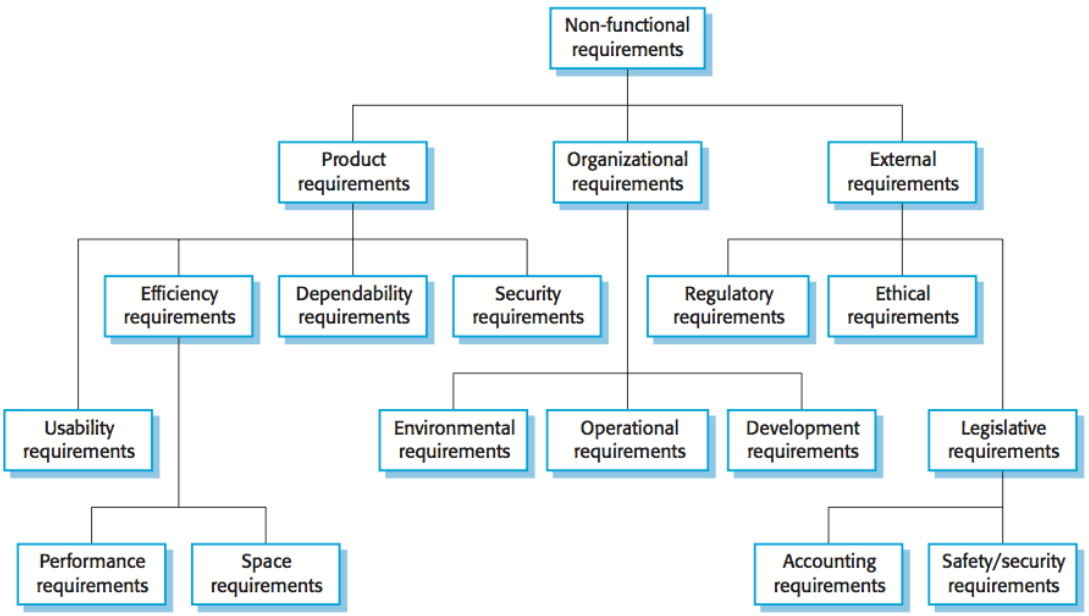
Requirements which specify that the delivered product must behave in a particular way, *e.g.* execution speed, reliability *etc.*

Organizational requirements

. Requirements which are a consequence of organizational 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.*



UNIT III SOFTWARE DESIGN

PART A

1. What are the elements of design model?
i. Data design
ii. Architectural design
iii. Interface design
iv. Component-level design

2. Define design process.
Design process is a sequence of steps carried through which the requirements are translated into a system or software model.

3. What is the benefit of modular design?
Changes made during testing and maintenance becomes manageable and they do not affect other modules.

4. What is a cohesive module?
A cohesive module performs only "one task" in software procedure with little interaction with other modules. In other words cohesive module performs only one thing.

5. What is Coupling?
Coupling is the measure of interconnection among modules in a program structure. It depends on the interface complexity between modules.

6. What is vertical partitioning?
Vertical partitioning often called factoring suggests that the control and work should be distributed top-down in program structure.

7.. What are the advantages of vertical partitioning?
i. These are easy to maintain changes.
ii. They reduce the change impact and error propagation.

8. What is Transform mapping?
The transform mapping is a set of design steps applied on the DFD in order to map the transformed flow characteristics into specific architectural style.

9. What is a Real time system?
Real time system is a software system in which the correct functionalities of the system are dependent upon results produced by the system and the time at which these results are produced.

10. What is SCM?
Software Configuration Management is a set of activities carried out for identifying, organizing and controlling changes throughout the lifecycle of computer software.

PART B

1. Explain in detail about Functional Modeling and Structural Modeling.

This model describes the computations that take place within a system. This model is useful when the transformation from the inputs to outputs is Complex. The functional model of a system can be represented by a data Flow Diagram (DFD).

Data Flow Diagrams/Data Flow Graph/Bubble chart

- A DFD is a graphical representation that depicts the information flow and the transforms that are applied as the data move from input to output.
- Level 0 DFD also called as fundamental system model or context model represents the entire software as a single bubble with input and output data indicated by incoming and outgoing arrows.
- Level 1 DFD contains 5 or 6 bubbles.
- Each bubbles can be refined at Layers to depict more details.

Structural Modeling.

- Structural model includes a detail refinement of ERD,data flow model and control flow model.
- Creating an ERD.
- Developing relationships and cardinality/Modality. Creating a data flow model using the guidelines.
- Creating a control flow model which describes the structural connection of
- Processes
- Control flows
- Control stores. State automation
- Process activation table.

2. What is the importance features of software architecture.

The fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution. Over time the set of significant design decisions about the organization of a software system which encompass:

(i) the selection of structural elements, their interfaces, and their collaborative behavior.
(ii) the composition of elements into progressively larger subsystems.
(iii) the architectural style that guides this organization.
(iv) system-level properties concerning usage, functionality, performance, resilience, reuse, constraints, trade- offs, and aesthetics.
(v) Roles of Architecture Description Shows the big picture about the software solution structures. Captures significant, early design decisions of a software system.
(vi) Enables to characterize or evaluate externally visible properties of the software system. (vii)Defines Constraints on the selection of implementation alternatives.
(viii)Serves as a skeleton around which full- fledged systems can be fleshed out.

3. Explain in details about Architectural styles.

Architectural styles.

Categories:

1. Components
2. Constraints
3. Connectors
4. Semantic models

Commonly used architectural styles are

1. Data centered architecture
2. Data flow architecture
3. Call and return architecture
4. Object oriented architecture
5. Layered architecture

Architectural patterns

1. Concurrency
2. Persistence
3. Distribution

4. Explain in details about Architectural

design Architectural designs

Representing system in context

- Target system
- Super ordinate systems
- Sub ordinate systems
- Actors
- Peer-level systems

Archetypes

- Point or node
- Control unit or controller
- Indicator or output

Refining architecture into components Instantiations of the system

5. Architectural Mapping using data flow

Transform flow
Transaction flow

Transform mapping

Step.1: Review fundamental system model to identify information flow
Step.2: Review and refine the data flow diagram for software
Step.3: Determine if the DFD has the transform or transaction flow characteristics **Step.4:**Isolate the transform center by specifyin incoming and outoing flow boundaries **Step.5:**Perform first level factoring
Step.6: Perform second level factoring
Step.7: Refine the first iteration achiecture using design heuristics for improved software quality

Transaction mapping

Step.1: Review fundamental system model to identify information flow
Step.2: Review and refine the data flow diagram for software
Step.3: Determine if the DFD has the transform or transaction flow characteristics
Step.4: Identify the transaction center and flow characteristics along each of the action paths **Step.5:**Map DFD into transaction processing structure
Step.6: Factor and refining the transaction structure and structure of each action path
Step.7: Refine the first iteration achiecture using design heuristics for improved software quality

6. Component Level Design:

Designing class based components Basic design principle

1. The open-closed principle
2. The liskov substitution principle
3. Dependency inversion principle
4. The interface segregation

principle Component level design guideline

- Components
- Interfaces
- Dependencies and interfaces

7. Expain in detail about cohesion and coupling

Cohesion

- Functional

the entity performs a single, well-defined task, without side effects.

- Layer
- Communicational
- Sequential
- Procedural
- Temporal cohesion - the entity is responsible for a set of tasks which must be performed at the same general time (e.g. initialization or cleanup)Utility
- Informational cohesion - the entity represents a cohesive body of data and a set of independent operations on that body of data.
- Logical - the entity is responsible for a set of related tasks, one of which is selected by the caller in each case.
- Utility cohesion - the entity is responsible for a set of related tasks which have no stronger form of cohesion.

Couplin g Types

- Content coupling - a module depends on the internal working of another module
- Common coupling - two modules share the same global data
- Control coupling - one module controls the flow of another, by passing it information on what to do
- Stamp coupling - modules share a composite data structure and use only part of it
- Data coupling - modules share data through, e.g., through parameters.
- External coupling

Traditional components

-Structured programming

- Graphical design notations
- **Tabular design notations**
- **Program design language(PDL)**

7. **Design Concepts-Design Model**

The design concepts provide the software designer with a foundation from which more sophisticated methods can be applied. A set of fundamental design concepts has evolved. They are:

1. Abstraction - Abstraction is the process or result of generalization by reducing the information

content of a concept or an observable phenomenon, typically in order to retain only information which is relevant for a particular purpose.
2. Refinement - It is the process of elaboration. A hierarchy is developed by decomposing a macroscopic statement of function in a step-wise fashion until programming language statements are reached. In each step, one or several instructions of a given program are decomposed into more detailed instructions. Abstraction and Refinement are complementary concepts.
3. Modularity - Software architecture is divided into components called modules.
4. Software Architecture - It refers to the overall structure of the software and the ways in which that structure provides conceptual integrity for a system. A good software architecture will yield a good return on investment with respect to the desired outcome of the project, e.g. in terms of performance, quality, schedule and cost.
5. Control Hierarchy - A program structure that represents the organization of a program component and implies a hierarchy of control.
6. Structural Partitioning - The program structure can be divided both horizontally and vertically. Horizontal partitions define separate branches of modular hierarchy for each major program function. Vertical partitioning suggests that control and work should be distributed top down in the program structure.
7. Data Structure - It is a representation of the logical relationship among individual elements of data.
8. Software Procedure - It focuses on the processing of each modules individually
9. Information Hiding - Modules should be specified and designed so that information contained within a module is inaccessible to other modules that have no need for such information

8. Design Heuristic

Heuristic evaluations are one of the most informal methods of usability inspection in the field of human-computer interaction. There are many sets of usability design heuristics; they are not mutually exclusive and cover many of the same aspects of user interface design.

Match between system and the real world:
The system should speak the user's language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

User control and freedom:
Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

Consistency and standards:
Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

Error prevention:
Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

Recognition rather than recall:
Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

Flexibility and efficiency of use:

Accelerators—unseen by the novice user—may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

Aesthetic and minimalist design:
Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

Help users recognize, diagnose, and recover from errors:
Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

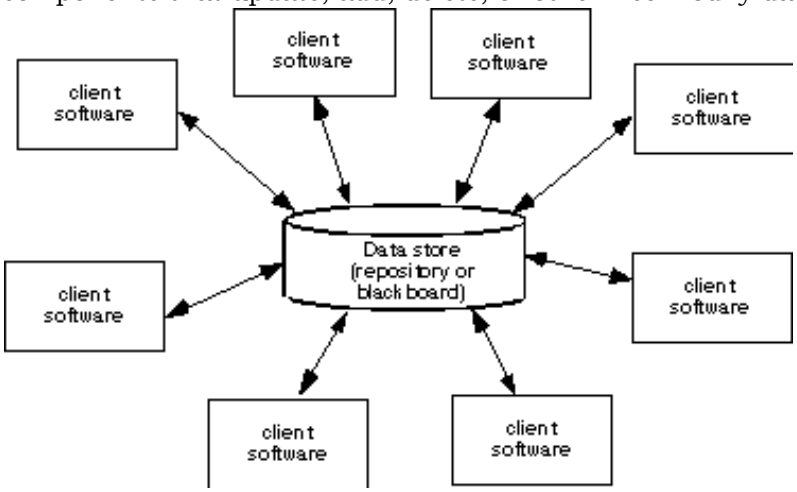
Help and documentation:
Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

9.Various Architecture in Architectural Styles

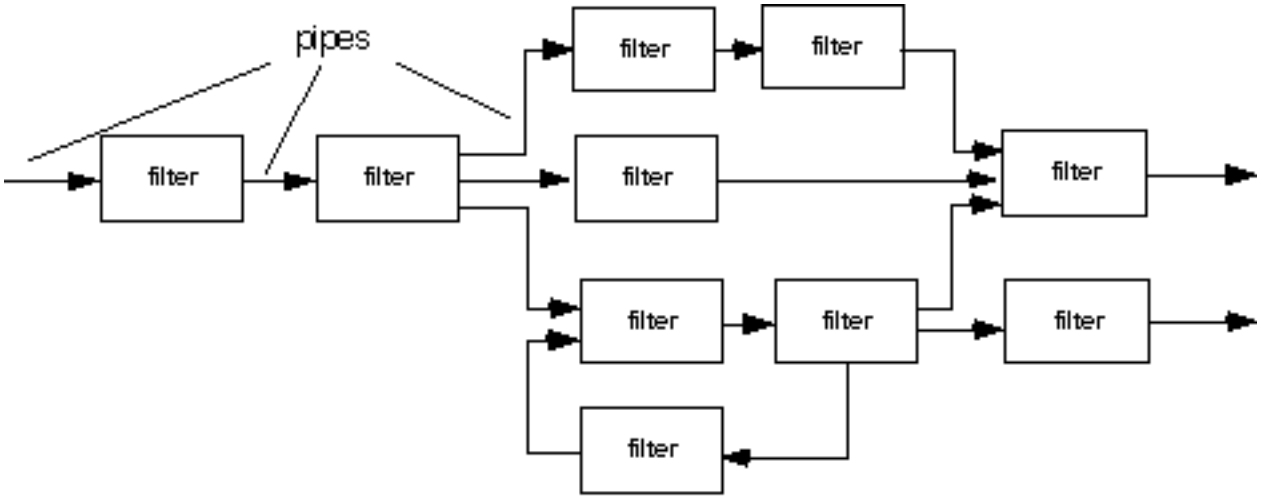
Each style describes a system category that encompasses:
(1) a set of components (e.g., a database, computational modules) that perform a function required by a system,
(2) a set of connectors that enable "communication, coordination and cooperation" among components,
(3) constraints that define how components can be integrated to form the system, and
(4) semantic models that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

- Data-centered architectures
- Data flow architectures
- Call and return architectures
- Object-oriented architectures
- Layered architectures

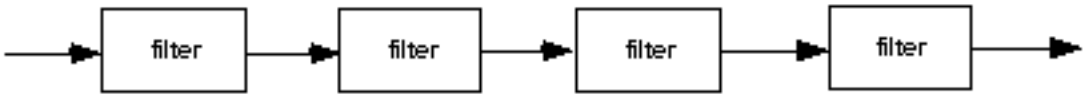
Data-Centered Architecture
A data store resides at the center of this architecture and is accessed frequently by other components that update, add, delete, or otherwise modify data within the store.



Data Flow Architecture

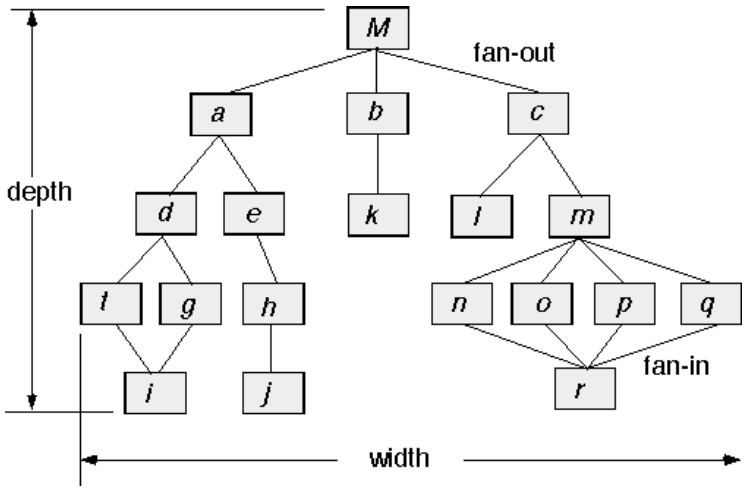


(a) pipes and filters

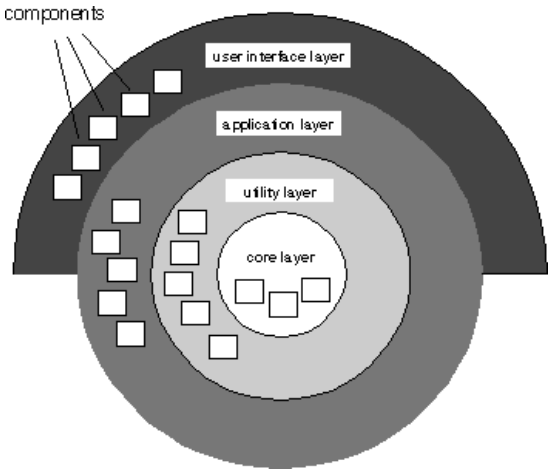


(b) batch sequential

Call and Return Architecture

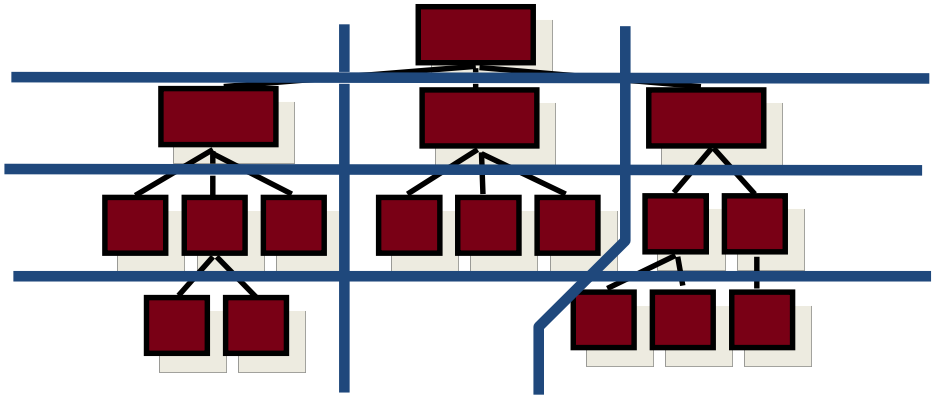


Layered Architecture



10. Partitioning the Architecture

- horizontal" and "vertical" partitioning are required



- define separate branches of the module hierarchy for each major function
- use control modules to coordinate communication between functions Vertical Partitioning:

Factoring

- design so that decision making and work are stratified
- decision making modules should reside at the top of the architecture Why Partitioned Architecture
- results in software that is easier to test
- leads to software that is easier to maintain

- results in propagation of fewer side effects
- results in software that is easier to extend

UNIT IV TESTING AND IMPLEMENTATION

PART A

1. Define software testing?

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design, and coding.

2. What are the objectives of testing?

i. Testing is a process of executing a program with the intend of finding an error.
ii. A good test case is one that has high probability of finding an undiscovered error. iii. A successful test is one that uncovers as an-yet undiscovered error.

3. Write short note on black box testing.

The black box testing is also called as behavioral testing. This method fully focuses on the functional requirements of the software. Tests are derived that fully exercise all functional requirements.

4. What is equivalence partitioning?

Equivalence partitioning is a black box technique that divides the input domain into classes of data. From this data test cases can be derived. Equivalence class represents a set of valid or invalid states for input conditions.

5. What are the various testing strategies for conventional software?

i. Unit testing
ii. Integration testing.
iii. Validation testing.
iv. System testing.

6. Define debugging.

Debugging is defined as the process of removal of defect. It occurs as a consequence of successful testing.

7. What are the common approaches in debugging?

 Brute force method: The memory dumps and run-time tracks are examined and program with write statements is loaded to obtain clues to error causes.
 Back tracking method: The source code is examined by looking backwards from symptom to potential causes of errors.
 Cause elimination method: This method uses binary partitioning to reduce the number of locations where errors can exists.

8. Distinguish between alpha and beta testing.

 Alpha and beta testing are the types of acceptance testing.
 Alpha test: The alpha testing is attesting in which the version of complete software is tested by the customer under the supervision of developer. This testing is performed at developer's site.
 Beta test: The beta testing is a testing in which the version of the software is tested by the customer without the developer being present. This testing is performed at customer's site.

9. What are the various types of system testing?

1. **Recovery testing** – is intended to check the system‟ s ability to recover from failures.
2. **Security testing –** verifies that system protection mechanism prevent improper penetration or data alteration.
3. **Stress testing –** Determines breakpoint of a system to establish maximum service level.
4. **Performance testing –** evaluates the run time performance of the software, especially real-time software.

10. What are the benefits of smoke testing?

- Integration risk is minimized.
- The quality of the end-product is improved.
- Error diagnosis and correction are simplified.
- Assessment of program is easy.

PART B

1. Explain about the software testing strategies.

A strategic approach to software

testing. Verification and Validation.

- Verification refers to the set of activities that ensure that software
- Correctly implements a specific function.

- Validation refers to a different set of activities that ensure that the software that has been built is traceable to the customer
- requirements. According to Boehm,

- Verification:" Are we building the product right?"

- Validation:" Are we building the right product?"

Organization for software testing A software testing strategy. Criteria for completion of testing.

2. Explain about integration testing? Integration testing.

It is a systematic technique for constructing the program structure.

The different types of integration techniques are

Incremental integration

The program is constructed and tested in small increments.

Top-down integration

It is an incremental approach.
Modules are integrated by moving downward through the control hierarchy beginning with the main control module(main program).
Subordinate modules are incorporated by depth-first or breadth-first manner.

Bottom-up integration

This testing begins construction and testing with the components at the lowest levels in the program structure.

Regression testing

It is the re-execution of some subset of tests that have already been conducted to ensure the changes that have not been propagated unintended side effects.

Smoke testing

It minimizes the integration risk.
Error diagnosis and correction are simplified

3. Explain in detail about black box testing and white box testing? White box testing:

1. condition
testing 2.loop
testing
-simple loop

-nested loop
-concatenated loop
-unstructured
loop 3.basis path
testing
Advantages:

- Each procedure can be tested thoroughly.
- It helps in optimizing the code
- White box testing can be easily automated

Black box testing:
Black box or behavioral testing focuses on the functional requirements of the software. It is applied during the last stage of testing.
Syntax driven testing is suitable for the specification which are described by a certain grammar.
Decision table based testing is implemented when the original software requirement have been formulated in the format of " if-then" statements.
Liquid level control is the study of a simple control problem which is designed to check the liquid level. It has 2 sensors.
1.equivalence
partitioning
2.boundary value
analysis Advantages:

- Incorrect or missing function
- Interface errors
- Errors in external data structures
- Performance errors
- Initialization or termination errors

4.Explain in details about testing strategy? (may-

05,06) Software Testing Types:

Black box testing – Internal system design is not considered in this type of testing. Tests are based on requirements and functionality.

White box testing – This testing is based on knowledge of the internal logic of an application's code. Also known as Glass box Testing. Internal software and code working should be known for this type of testing. Tests are based on coverage of code statements, branches, paths, conditions.

Unit testing – Testing of individual software components or modules. Typically done by the programmer and not by testers, as it requires detailed knowledge of the internal program design and code. may require developing test driver modules or test harnesses.

Incremental integration testing – Bottom up approach for testing i.e continuous testing of an application as new functionality is added; Application functionality and modules should be independent enough to test separately. done by programmers or by testers.

Integration testing – Testing of integrated modules to verify combined functionality after integration. Modules are typically code modules, individual applications, client and server applications on a network, etc. This type of testing is especially relevant to client/server and distributed systems.

Functional testing – This type of testing ignores the internal parts and focus on the output is as per requirement or not. Black-box type testing geared to functional requirements of an application.

System testing – Entire system is tested as per the requirements. Black-box type testing that is based on overall requirements specifications, covers all combined parts of a system.

End-to-end testing – Similar to system testing, involves testing of a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate.

Sanity testing - Testing to determine if a new software version is performing well enough to accept it for a major testing effort. If application is crashing for initial use then system is not stable enough for further testing and build or application is assigned to fix.

Regression testing – Testing the application as a whole for the modification in any module or functionality. Difficult to cover all the system in regression testing so typically automation tools are used for these testing types.

Acceptance testing -Normally this type of testing is done to verify if system meets the customer specified requirements. User or customer do this testing to determine whether to accept application.

Load testing – Its a performance testing to check system behavior under load. Testing an application under heavy loads, such as testing of a web site under a range of loads to determine at what point the system's response time degrades or fails.

Stress testing – System is stressed beyond its specifications to check how and when it fails. Performed under heavy load like putting large number beyond storage capacity, complex database queries, continuous input to system or database load.

Performance testing – Term often used interchangeably with 'stress' and 'load' testing. To check whether system meets performance requirements. Used different performance and load tools to do this.

Usability testing – User-friendliness check. Application flow is tested, Can new user understand the application easily, Proper help documented whenever user stuck at any point. Basically system navigation is checked in this testing.

Install/uninstall testing - Tested for full, partial, or upgrade install/uninstall processes on different operating systems under different hardware, software environment.

Recovery testing – Testing how well a system recovers from crashes, hardware failures, or other catastrophic problems.

Security testing – Can system be penetrated by any hacking way. Testing how well the system protects against unauthorized internal or external access. Checked if system, database is safe from external attacks.

Compatibility testing – Testing how well software performs in a particular hardware/software/operating system/network environment and different combination s of above.

Comparison testing – Comparison of product strengths and weaknesses with previous versions or other similar products.

Alpha testing – In house virtual user environment can be created for this type of testing. Testing is done at the end of development. Still minor design changes may be made as a result of such testing.

Beta testing – Testing typically done by end-users or others. Final testing before releasing application for commercial purpose

5. Explain about coding practices in software implementation techniques?

coding practices:

- control construct
- use of gotos
- information hiding
- nesting
- user defined data types
- module size
- module interface
- side effects
- robustness
- switch case with defaults
- empty catch block
- empty if and while statement
- check for read return
- return from finally block
- trusted data sources
- correlated parameters
- exceptions handling

6. Expalin about coding standards and refactoring in detail?

Coding standards:

- Naming conventions
- Files
- Commenting/layout
- Statemen

ts Refactoring:

Refactoring is defined as change mode to the internal structure of software for better understanding and performing cheaper to modification without changing system's behavior.

7. Explain in detail about debugging?

Debugging is a process of removal of a defect. It occurs as a consequence of successful testing Common approaches in debugging are:

- Brute force method

  The memory dumps and runtime traces are examined and program with write statements is loaded to obtain clues to errors
- Backtracking method

  This method is applicable to small programs. In this method, the source is examined by looking backwards from symptom to potential causes of errors.

- Cause elimination method

  This method uses binary partitioning to reduce the number of locations where errors can exist.

UNIT V PROJECT MANAGEMENT
PART A

1. Define measure.
Measure is defined as a quantitative indication of the extent, amount, dimension, or size of some attribute of a product or process.

2. Define metrics.
Metrics is defined as the degree to which a system component, or process possesses a given attribute.

3. What are the types of metrics?
_ **Direct metrics –** It refers to immediately measurable attributes. Example – Lines of code, execution
□ speed.
Indirect metrics – It refers to the aspects that are not immediately quantifiable or measurable. Example – functionality of a program.

4. What is COCOMO model?
COnstructive COst MOdel is a cost model, which gives the estimate of number of man-months it will take to develop the software product.

5.What is the purpose of timeline chart?
The purpose of the timeline chart is to emphasize the scope of the individual task. Hence set of tasks are given as input to the timeline chart.

6. What is EVA?
Earned Value Analysis is a technique of performing quantitative analysis of the software Project. It provides a common value scale for every task of software project. It acts as a measure for software project progress.

7.What is software maintenance?
Software maintenance is an activity in which program is modified after it has been put into use.

8. Define maintenance.
Maintenance is defined as the process in which changes are implemented by either modifying the existing system‟ s architecture or by adding new components to the system.

9.What is architectural evolution?
Architectural evolution is the process of changing a system from a centralized architecture to a distributed architecture like client server.

10. How the CASE tools are classified?
CASE tools can be classified by a. By function or use
b. By user type(e.g. manager,tester),or
c. By stage in software engineering process (e.g.requirements,test).

PART B

1. **Write a note on i)Cocomo model ii)SOFTWARE METRICS**

i)COCOMO estimation
criteria Types
Three
classes
Merits
Limitations
Example
ii)Software metrics
Size oriented
approach
Function oriented approach

2. **What are the needs for software measures**

Need for software
measures Quantitative
indication Amount
Dimensio
n Process

3. **Write a note task scheduling**

Task scheduling
Relationship between people and
effort Task sets
Task layouts
Time line charts
Tracking
schedule

4. **Explain Direct measures**

Loc based estimation
The problems of lines of code (LOC)
– Different languages lead to different
lengths of code
– It is not clear how to count lines of code
– A report, screen, or GUI generator can
generate thousands of lines of code in
minutes
– Depending on the application, the
complexity of code is different

Advantages

Disadvantag

es
Example of LOC based estimation

5. **Explain Function Point**
analysis Function Point
Analysis

Software systems, unless they are thoroughly understood, can be like an ice berg. They are becoming more and more difficult to understand. Improvement of coding tools allows software developers to produce large amounts of software to meet an ever expanding need from users. As systems grow a method to understand and communicate size needs to be used. Function Point Analysis is a structured technique of problem solving. It is a method to break systems into smaller components, so they can be better understood and analyzed.
Characteristic of Quality Function Point Analysis

Function Point Analysis should be performed by trained and experienced personnel. If Function Point Analysis is conducted by untrained personnel, it is reasonable to assume the analysis will done incorrectly. The personnel counting function points should utilize the most current version of the Function Point Counting Practices Manual,

The Five Major Components

External Inputs (EI)
External Outputs
(EO) External
Inquiry (EQ)
Internal Logical Files (ILF's)
External Interface Files
(EIF's)

Type of Component	Complexity of Components			
	Low	Average	High	Total
External Inputs	___ x 3 = ___	___ x 4 = ___	___ x 6 = ___	
External Outputs	___ x 4 = ___	___ x 5 = ___	___ x 7 = ___	
External Inquiries	___ x 3 = ___	___ x 4 = ___	___ x 6 = ___	
Internal Logical Files	___ x 7 = ___	___ x 10 = ___	___ x 15 = ___	
External Interface Files	___ x 5 = ___	___ x 7 = ___	___ x 10 = ___	
		Total Number of Unadjusted Function Points		
		Multiplied Value Adjustment Factor		
		Total Adjusted Function Points		

Benefits of Function Point Analysis
Function Points can be used to size software applications accurately. Sizing is an important component in determining productivity (outputs/inputs).
They can be counted by different people, at different times, to obtain the same measure within a reasonable margin of error.
Function Points are easily understood by the non technical user. This helps communicate sizing information to a user or customer.
Function Points can be used to determine whether a tool, a language, an environment, is more productive when compared with others.

6. Write a note on Risk management.

Software risks:

- What can go wrong?
- What is the likelihood?
- What will be the damage?
- What can be done about it?

Risk analysis and management are a set of activities that help a software team to understand and manage uncertainty about a project.

Risk is the uncertainty associated with the outcome of a future event and has a number of attributes:

- Uncertainty (probability)
- Time (future event)
- Potential for loss (or gain)
- Multiple perspectives, e.g.,
- Process perspective (development process breaks)
- Project perspective (critical objectives are missed)
- Product perspective (loss of code integrity)

- User perspective (loss of functionality

The process by which a course of action is selected that balances the potential impact of a risk weighted by its probability of occurrence and the benefits of avoiding (or controlling) the risk

Risk management life cycle:

- Identify (risk identification)
- Analyze (risk analysis)
- Plan (contingency planning)
- Track (risk monitoring)
- Control (recovery management)

Risk Projection

Estimate the probability of occurrence
Estimate the impact on the project on a particular scale, e.g.,

- low impact (negligible)
- medium impact (marginal)
- high impact (critical)
- very high impact (catastrophic)

Build a risk table and sort by probability and impact

Definition
Software
risks Types
of risks
Project risks
Technical risks
Business risks
Predictable risks
Unpredictable
risks
Reactive and proactive
risks Risk identification
Risk projection

7. **Write a note a RMMM.**
    - Risk mitigation
    - Risk monitoring
    - Risk management
    - RMMM plan

Risk: Computer Crash

· **Mitigation**

The cost associated with a computer crash resulting in a loss of data is crucial. A computer crash itself is not crucial, but rather the loss of data. A loss of data will result in not being able to deliver the product to the customer. This will result in a not receiving a letter of acceptance from the customer. Without the letter of acceptance, the group will receive a failing grade for the course. As a result the organization is taking steps to make multiple backup copies of the software in
development and all documentation associated with it, in multiple locations.

· **Monitoring**

When working on the product or documentation, the staff member should always be aware of the stability of the computing environment they're working in. Any changes in the stability of the environment should be recognized and taken seriously.

· **Management**

The lack of a stable-computing environment is extremely hazardous to a software development team. In the event that the computing environment is found unstable, the development team should cease work on that system until the environment is made stable again, or should move to a system that is stable and continue working there.

Risk: Technology Does Not Meet Specifications

· **Mitigation**

In order to prevent this from happening, meetings (formal and informal) will be held with the customer on a routine business. This insures that the product we are producing, and the specifications of the customer are equivalent.

· **Monitoring**

The meetings with the customer should ensure that the customer and our organization understand each other and the requirements for the product.

· **Management**

Should the development team come to the realization that their idea of the product specifications differs from those of the customer, the customer should be immediately notified and whatever steps necessary to rectify this problem should be done. Preferably a meeting should be held between the development team and the customer to discuss at length this issue.

Risk: Changes in Requirements

· **Mitigation**

In order to prevent this from happening, meetings (formal and informal) will be held with the customer on a routine business. This insures that the product we are producing, and the requirements of the customer are equivalent.

· **Monitoring**

The meetings with the customer should ensure that the customer and our organization understand each other and the requirements for the product.

· **Management**

Should the development team come to the realization that their idea of the product requirements differs from those of the customer, the customer should be immediately notified and whatever steps necessary to rectify this problem should be taken. Preferably a meeting should be held between the development team and the customer to discuss at length this issue.