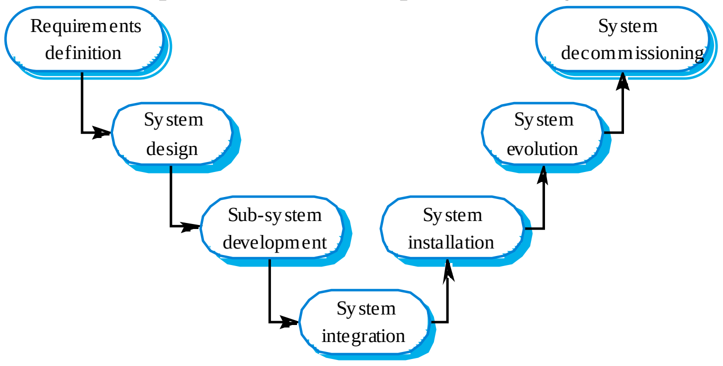
Q.2 a. What is System Engineering? Explain in detail. (12)

Answer:

Systems engineering is the activity of specifying, designing, implementing, validating, deploying and maintaining socio – technical systems.

Concerned with the services provided by the system, constraints on its construction and operation and the ways in which it is used.

Systems Engineering Process: Usually follows a 'waterfall' model because of the need for parallel development of different parts of the system



the <u>System Requirement definition</u>: specifies what the system should do and its essential and desirable system properties. Concentrates on deriving three types of requirements:

1. Abstract functional requirements.
2. System properties
3. Characteristics that the system must not exhibit.

<u>System Design</u>: Activities involved in this process are:

1. Partition requirements
2. Identify sub – systems
3. Assign requirements to sub system
4. Specify sub – system functionality
5. Define Sub – system interfaces

<u>System Modeling</u>: system requirements and design activity, systems may be modeled as a set of components and relationships between these components.

An architectural model presents an abstract view of the sub-systems making up a system

May include major information flows between sub-systems

Usually presented as a block diagram

May identify different types of functional component in the model

<u>Sub – system Development:</u> The identified sub – systems during system design are implemented . Involves starting another system engineering process for individual sub – systems. Sub – Systems are developed from scratch or bought off the shelf and integrated into the system. Usually sub – systems are developed in parallel. When problems encountered a system modification request is made.

<u>System Integration:</u> Independently developed sub –systems are put together to form a complete system. For the following reasons the sub – systems are integrate one at a time.

1. Impossible to schedule the development of all the sub –systems are not possible.
2. Reduces the cost of error location.

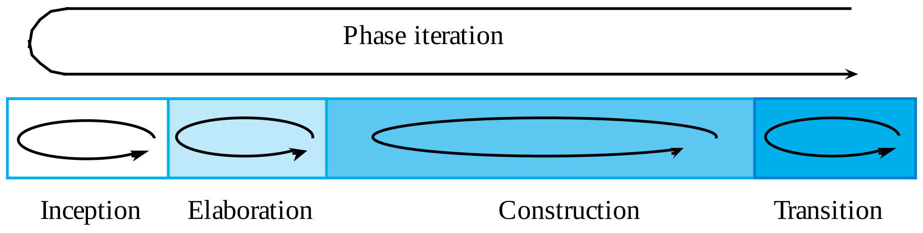
System Evolution:

System Decommissioning

b. Explain Rational Unified Process (RUP). (4)

Answer:

A modern process model derived from the work on the UML and associated process.
Normally described from 3 perspectives
A dynamic perspective that shows phases over time;
A static perspective that shows process activities;
A practive perspective that suggests good practice.



Inception: Establish the business case for the system.
Elaboration: Develop an understanding of the problem domain and the system architecture.
Construction: System design, programming and testing.
Transition: Deploy the system in its operating environment.
Good Practices: Develop software iteratively
Manage requirements
Use component-based architectures
Visually model software
Verify software quality
Control changes to software

Q.3 a. Discuss in detail Functional and Nonfunctional Requirements. (12)

Answer:

Functional Requirements:

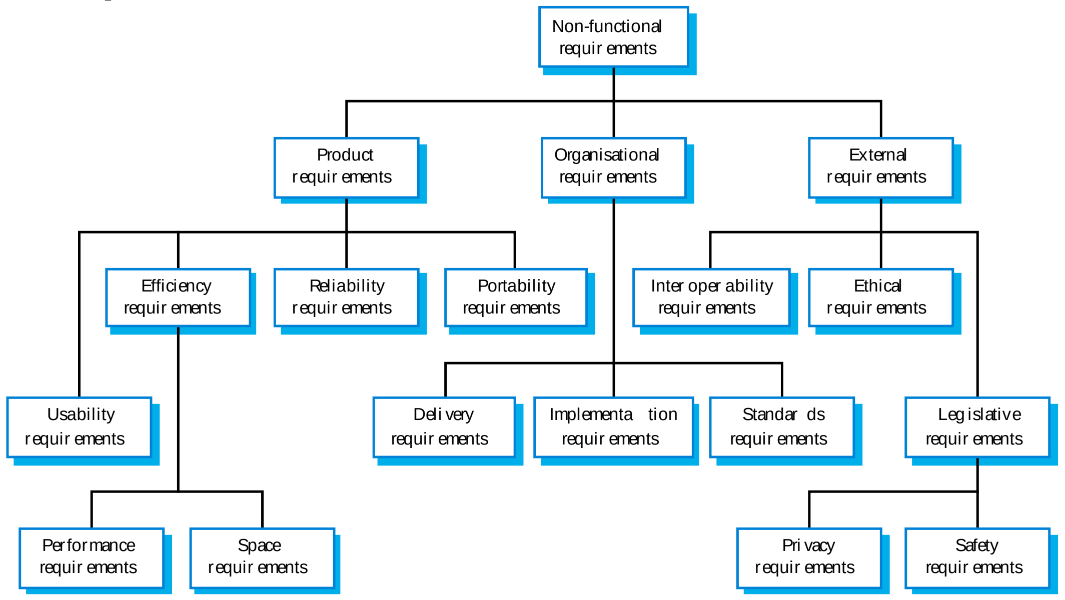
Describe functionality or system services.
Depend on the type of software, expected users and the type of system where the software is used.
Functional user requirements may be high-level statements of what the system should do but functional system requirements should describe the system services in detail.
Examples of Functional Requirements:
A library system that provides a single interface to a number of databases of articles in different libraries.
Users can search for, download and print these articles for personal study
The user shall be able to search either all of the initial set of databases or select a subset from it.
The system shall provide appropriate viewers for the user to read documents in the document store.
Every order shall be allocated a unique identifier (ORDER_ID) which the user shall be able to copy to the account's permanent storage area
In principle, requirements should be both complete and consistent.
Complete: They should include descriptions of all facilities required.
Consistent: There should be no conflicts or contradictions in the descriptions of the system facilities.
In practice, it is impossible to produce a complete and consistent requirements document.

Nonfunctional Requirements: These define system properties and constraints e.g. reliability,

response time and storage requirements. Constraints are I/O device capability, system representations, etc.
Process requirements may also be specified mandating a particular CASE system, programming language or development method.
Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless.

Classifications:

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.



Non-functional requirements may be very difficult to state precisely and imprecise requirements may be difficult to verify.
Goal: A general intention of the user such as ease of use.
Verifiable non-functional requirement: A statement using some measure that can be objectively tested.

Goals are helpful to developers as they convey the intentions of the system users.

Property	Measure
Speed	Processed transactions/secondUser/ Event response time, Screen refresh time
Size	M BytesNumber of ROM 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

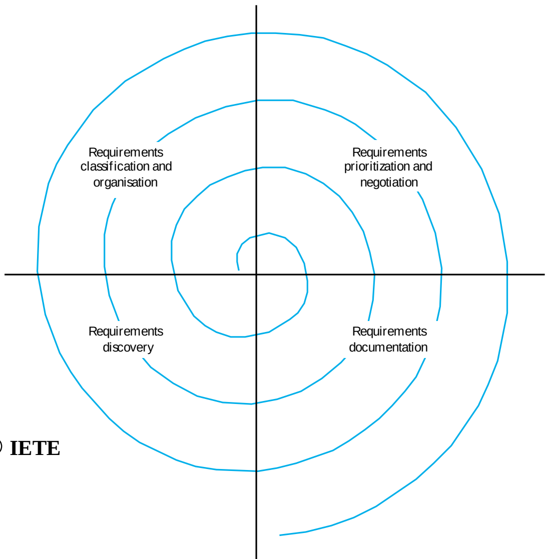
Requirements Interaction: Conflicts between different non-functional requirements are common in complex systems.
Spacecraft system:

- To minimise weight, the number of separate chips in the system should be minimised.
- To minimise power consumption, lower power chips should be used.
- However, using low power chips may mean that more chips have to be used. Which is the most critical requirement?

b. Discuss about Requirement Elicitation and Analysis phase. (4)

Answer:
Sometimes called requirements elicitation or requirements discovery.
Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints.
May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called *stakeholders.*
Problems in Requirement Analysis Phase:

Stakeholders don't know what they really want.

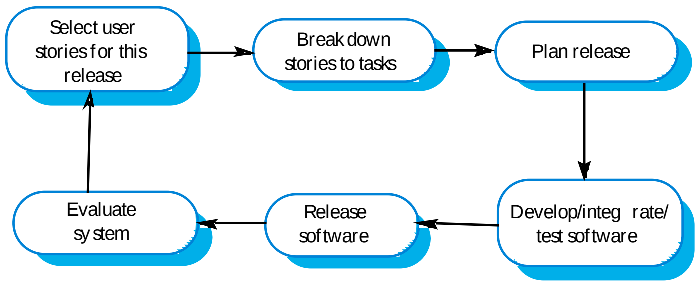

Requirements classification and organisation
Requirements prioritization and negotiation
Requirements discovery
Requirements documentation

Stakeholders express requirements in their own terms.
Different stakeholders may have conflicting requirements.
Organisational and political factors may influence the system requirements.
The requirements change during the analysis process. New stakeholders may emerge and the business environment change.
Requirements discovery: Interacting with stakeholders to discover their requirements. Domain

requirements are also discovered at this stage.
Requirements classification and organisation: Groups related requirements and organises them into coherent clusters.
Prioritisation and negotiation: Prioritising requirements and resolving requirements conflicts.
Requirements documentation: Requirements are documented and input into the next round of the spiral.

Q.4 a. Explain in detail about Extreme Programming and explain how testing is done in Extreme Programming (10)

Answer:
Perhaps the best-known and most widely used agile method.
Extreme Programming (XP) takes an 'extreme' approach to iterative development.
New versions may be built several times per day;
Increments are delivered to customers every 2 weeks;
All tests must be run for every build and the build is only accepted if tests run successfully


Select user stories for this release
Break down stories to tasks
Plan release
Develop/integ rate/ test software
Release software
Evaluate system

Principle or Practice	Description
Incremental planning	Requirements are recorded on Story Cards and the Stories to be included in a release are determined by the time available and their relative priority. The developers break these Stories into development 'Tasks'.
Small Releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple Design	Enough design is carried out to meet the current requirements and no more.
Test first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.
Pair Programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective Ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers own all the code. Anyone can change anything.

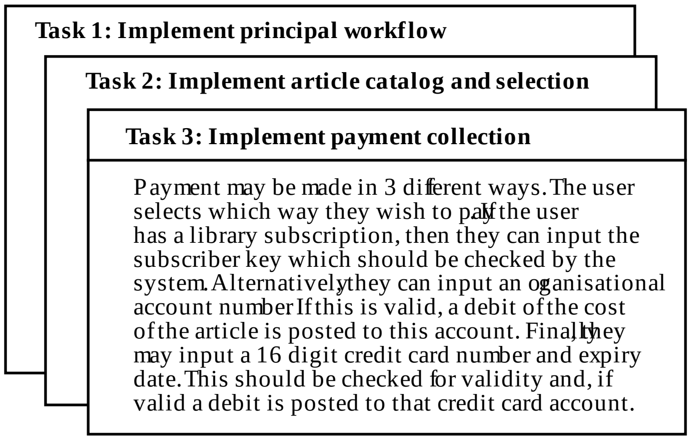
Principle or Practice	Description
Continuous Integration	As soon as work on a task is complete it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of over-time are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site Customer	A representative of the end-user of the system (the Customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

<u>Testing in XP:</u>

- Test-first development.
- Incremental test development from scenarios.
- User involvement in test development and validation.
- Automated test harnesses are used to run all component tests each time that a new release is built.

Testing card

- Writing tests before code clarifies the requirements to be implemented.
- Tests are written as programs rather than data so that they can be executed automatically. The test includes a check that it has executed correctly.
- All previous and new tests are automatically run when new functionality is added. Thus checking that the new functionality has not introduced errors



Task 1: Implement principal workflow

Task 2: Implement article catalog and selection

Task 3: Implement payment collection

Payment may be made in 3 different ways. The user selects which way they wish to pay. If the user has a library subscription, then they can input the subscriber key which should be checked by the system. Alternatively, they can input an organisational account number. If this is valid, a debit of the cost of the article is posted to this account. Finally, they may input a 16 digit credit card number and expiry date. This should be checked for validity and, if valid a debit is posted to that credit card account.

Pair programming:

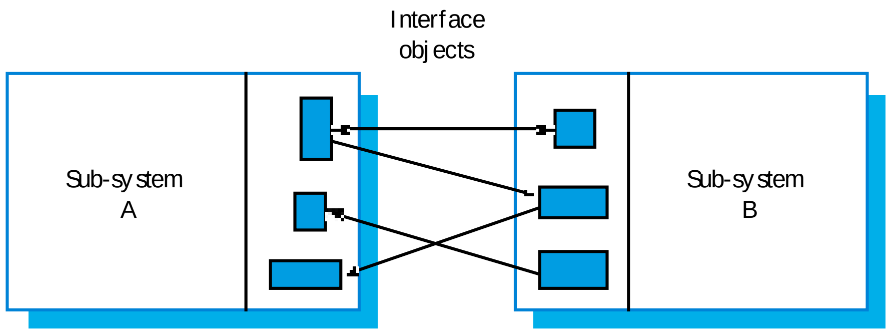
- In XP, programmers work in pairs, sitting together to develop code.
- This helps develop common ownership of code and spreads knowledge across the team.
- It serves as an informal review process as each line of code is looked at by more than 1 person.
- It encourages refactoring as the whole team can benefit from this.
- Measurements suggest that development productivity with pair programming is similar to that of two people working independently.

b. Discuss Sub-System Interface Specification. (6)

Answer:

<u>Sub System Interface Specification:</u>

- Large systems are decomposed into subsystems with well-defined interfaces between these subsystems.
- Specification of subsystem interfaces allows independent development of the different subsystems.
- Interfaces may be defined as abstract data types or object classes.
- The algebraic approach to formal specification is particularly well-suited to interface specification as it is focused on the defined operations in an object.



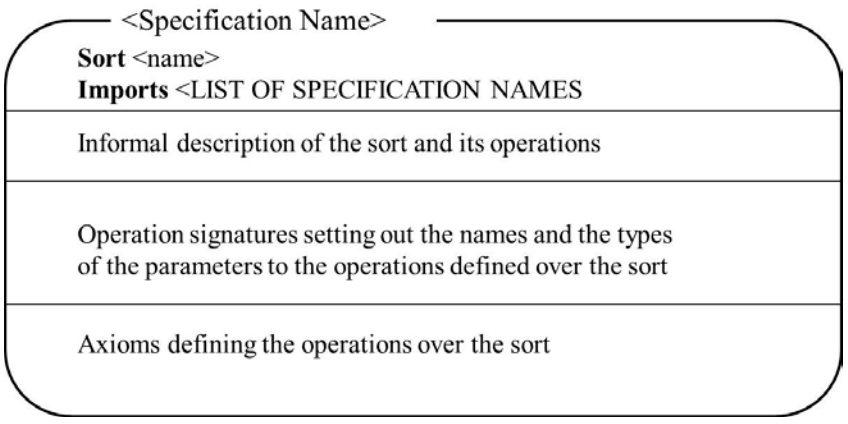
Structure of Algebraic Specification:

- o Introduction – Defines the sort (the type name) and declares other specifications that are used.
- o Description – Informally describes the operations on the type.
- o Signature – Defines the syntax of the operations in the interface and their parameters.

- Axioms
  - o Defines the operation semantics by defining axioms which characterise behaviour

The process of developing specification for sub – system interface includes the following activities:

- Specification structuring;
- Specification naming;
- Operation selection;
- Informal operation specification;
- Syntax definition;
- Axiom definition.



<Specification Name>

Sort <name>
Imports <LIST OF SPECIFICATION NAMES

Informal description of the sort and its operations

Operation signatures setting out the names and the types of the parameters to the operations defined over the sort

Axioms defining the operations over the sort

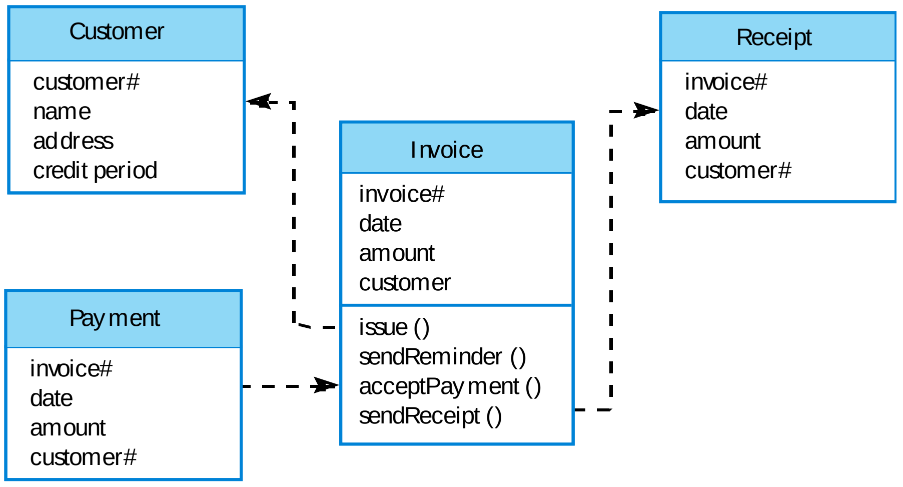
Q.5 a. Discuss Object Oriented style of modular decomposition (4)

Answer:

- Another structural level where sub-systems are decomposed into modules.
- Two modular decomposition models covered
  - <u>An object model</u> where the system is decomposed into interacting object;
  - <u>A pipeline or data-flow model</u> where the system is decomposed into functional modules which transform inputs to outputs.

<u>Object oriented decomposition:</u>

- If possible, decisions about concurrency should be delayed until modules are implemented.
- Structure the system into a set of loosely coupled objects with well-defined interfaces.
- Object-oriented decomposition is concerned with identifying object classes, their attributes and operations.
- When implemented, objects are created from these classes and some control model used to coordinate object operations.



b. Explain Distributed Systems Architecture in detail. (12)

Answer:

<u>Distributed system Characteristics:</u>

- ➢ Resource sharing - Sharing of hardware and software resources.
- ➢ Openness - Use of equipment and software from different vendors.
- ➢ Concurrency - Concurrent processing to enhance performance.
- ➢ Scalability - Increased throughput by adding new resources.
- ➢ Fault tolerance - The ability to continue in operation after a fault has occurred.

<u>Distributed Systems Architectures:</u>

- Client-server architectures

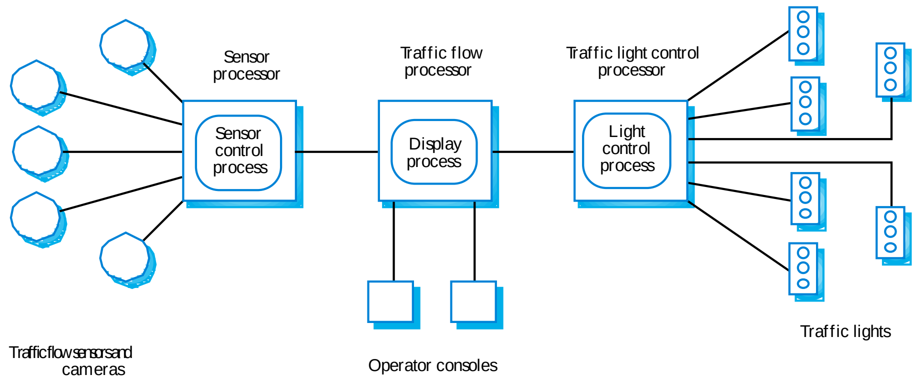
  Distributed services which are called on by clients. Servers that provide services are treated differently from clients that use services.
- Distributed object architectures

  No distinction between clients and servers. Any object on the system may provide and use services from other objects

<u>Multi-Processor Archictecture:</u>

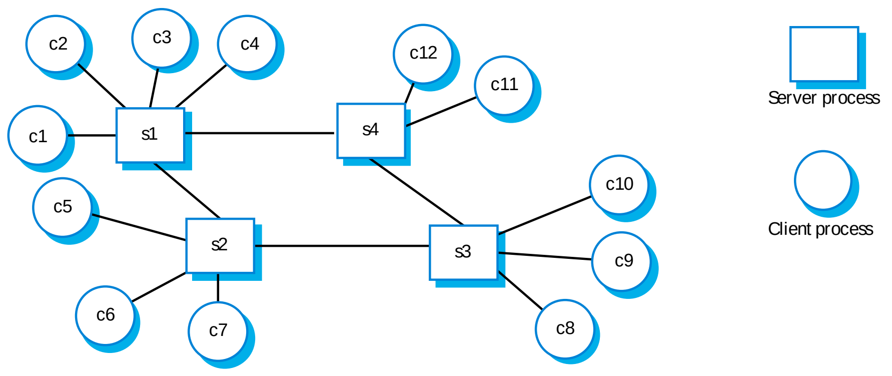
- Simplest distributed system model.

- System composed of multiple processes which may (but need not) execute on different processors.
- Architectural model of many large real-time systems.
- Distribution of process to processor may be pre-ordered or may be under the control of a dispatcher.



Client Server Architecture:

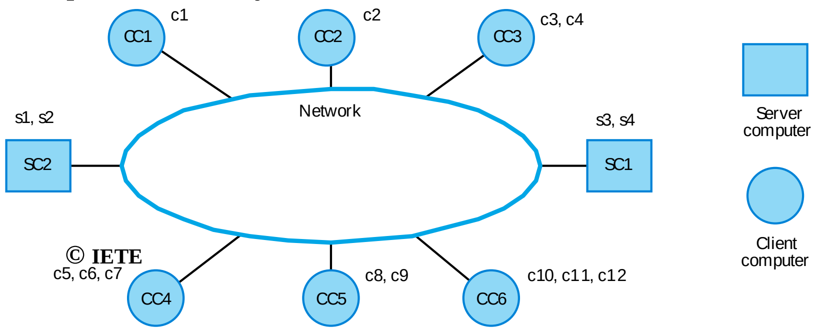
- The application is modelled as a set of services that are provided by servers and a set of clients that use these services.
- Clients know of servers but servers need not know of clients.
- Clients and servers are logical processes
- The mapping of processors to processes is not necessarily 1 : 1.



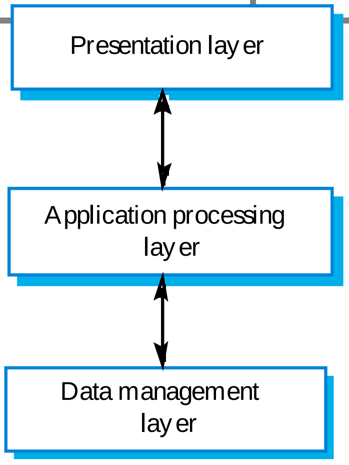
Layered Application Architecture:

- o Presentation layer – Concerned with presenting the results of a computation to system users and with collecting user inputs.
- o Application processing layer – Concerned with providing application specific functionality e.g., in a banking system, banking functions such as open account, close account, etc.
- o Data management layer – Concerned with managing the system databases.

Computers in C/S system:



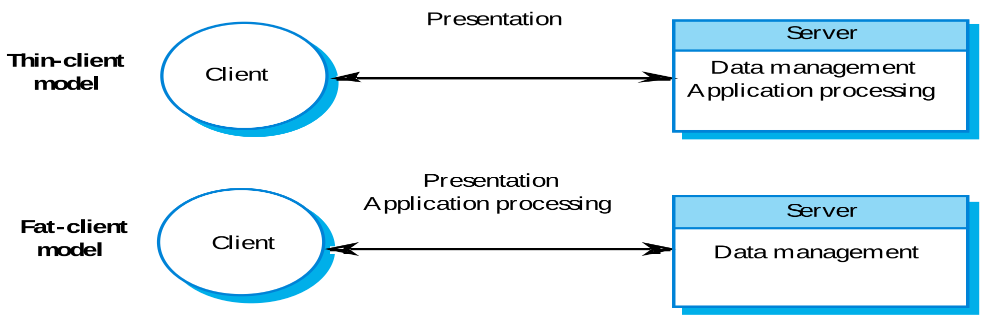
Layered Architecture:



- Presentation layer
    - Concerned with presenting the results of a computation to system users and with collecting user inputs.
- Application processing layer
    - Concerned with providing application specific functionality e.g., in a banking system, banking functions such as open account, close account, etc.
- Data management layer
    - Concerned with managing the system databases.

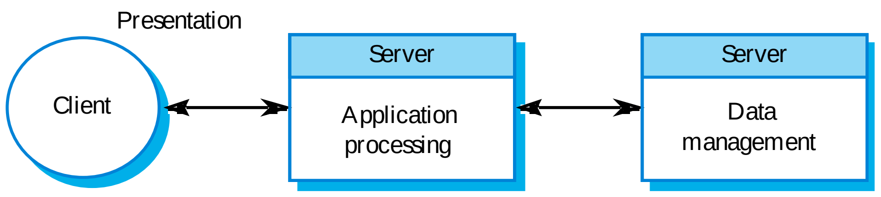
Thin and Fat clients:

- Thin-client model
    - In a thin-client model, all of the application processing and data management is carried out on the server. The client is simply responsible for running the presentation software.
- Fat-client model
    - In this model, the server is only responsible for data management. The software on the client implements the application logic and the interactions with the system user.



Three Tier Architecture:

- In a three-tier architecture, each of the application architecture layers may execute on a separate processor.
- Allows for better performance than a thin-client approach and is simpler to manage than a fat-client approach.
- A more scalable architecture - as demands increase, extra servers can be added.



Q.6 a. What is Object Oriented Design Process? Explain the first three stages of the process. (12)

Answer:

- Structured design processes involve developing a number of different system models.

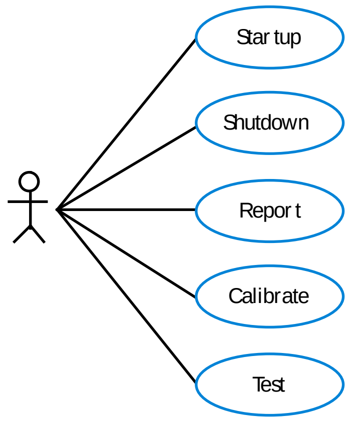
- They require a lot of effort for development and maintenance of these models and, for small systems, this may not be cost-effective.
- However, for large systems developed by different groups design models are an essential communication mechanism.

Various stages
- Define the context and modes of use of the system;
- Design the system architecture;
- Identify the principal system objects;
- Develop design models;
- Specify object interfaces.

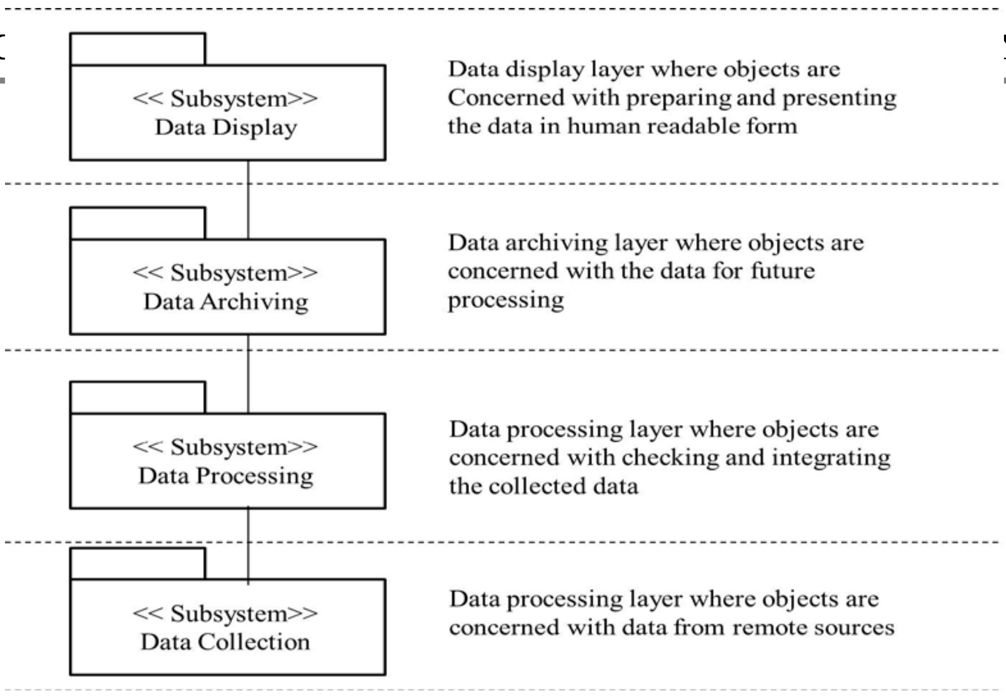
Defining System context and modes of the system:
- Develop an understanding of the relationships between the software being designed and its external environment
- System context
  - A static model that describes other systems in the environment. Use a subsystem model to show other systems. Following slide shows the systems around the weather station system.
- Model of system use
  - A dynamic model that describes how the system interacts with its environment. Use use-cases to show interactions

Use Case Models:
- Use-case models are used to represent each interaction with the system.
- A use-case model shows the system features as ellipses and the interacting entity as a stick figure

Architectural Design:
- Once interactions between the system and its environment have been understood, you use this information for designing the system architecture.



- A layered architecture is appropriate for the weather station
  - Interface layer for handling communications;
  - Data collection layer for managing instruments;
  - Instruments layer for collecting data.
- There should normally be no more than 7 entities in an architectural model


<< Subsystem>>
Data Display
Data display layer where objects are Concerned with preparing and presenting the data in human readable form
<< Subsystem>>
Data Archiving
Data archiving layer where objects are concerned with the data for future processing
<< Subsystem>>
Data Processing
Data processing layer where objects are concerned with checking and integrating the collected data
<< Subsystem>>
Data Collection
Data processing layer where objects are concerned with data from remote sources

Object Identification:

- Identifying objects (or object classes) is the most difficult part of object oriented design.
- There is no 'magic formula' for object identification. It relies on the skill, experience and domain knowledge of system designers.
- Object identification is an iterative process. You are unlikely to get it right first time.

Approaches to identification:

- Use a grammatical approach based on a natural language description of the system (used in Hood OOD method).
- Base the identification on tangible things in the application domain.
- Use a behavioural approach and identify objects based on what participates in what behaviour.
- Use a scenario-based analysis. The objects, attributes and methods in each scenario are identified.

b. Explain Generator based Reuse. (4)

Answer:

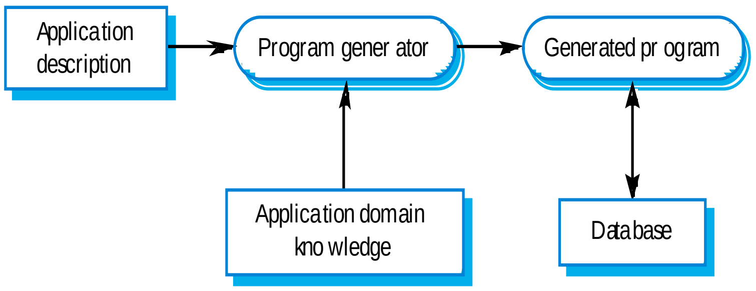
- Program generators involve the reuse of standard patterns and algorithms.
- These are embedded in the generator and parameterised by user commands. A program is then automatically generated.
- Generator-based reuse is possible when domain abstractions and their mapping to executable code can be identified.
- A domain specific language is used to compose and control these abstractions.

Types of Program Generators

- Application generators for business data processing;
- Parser and lexical analyser generators for language processing;
- Code generators in CASE tools.

- Generator-based reuse is very cost-effective but its applicability is limited to a relatively small number of application domains.

It is easier for end-users to develop programs using generators compared to other component-based approaches to reuse.



Q.7 a. Explain the User Interface design process. (12)

Answer:

An iterative process where users interact with the designers and interface prototypes to decide upon the features, organization and the look and feel of the system user interface.

The 3 core activities in this process are:

- User analysis. Understand what the users will do with the system;
- System prototyping. Develop a series of prototypes for experiment;
- Interface evaluation. Experiment with these prototypes with users.

User Analysis:

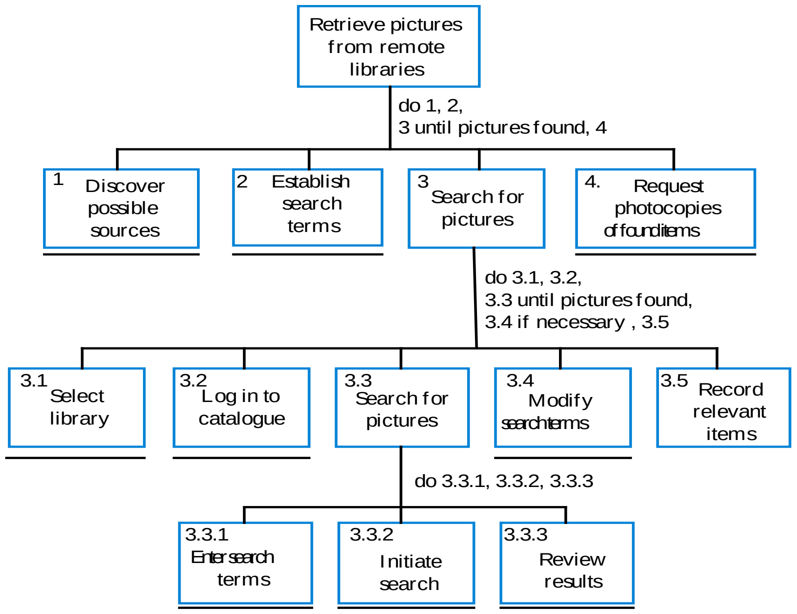
- If you don't understand what the users want to do with a system, you have no realistic prospect of designing an effective interface.
- User analyses have to be described in terms that users and other designers can understand.
- Scenarios where you describe typical episodes of use, are one way of describing these analyses.

Requirements from the scenario:

- Users may not be aware of appropriate search terms so need a way of helping them choose terms.
- Users have to be able to select collections to search.
- Users need to be able to carry out searches and request copies of relevant material.

Analysis Technique:

- Task analysis – Models the steps involved in completing a task.
- Interviewing and questionnaires –Asks the users about the work they do.
- Ethnography – Observes the user at work.


Retrieve pictures from remote libraries
do 1, 2, 3 until pictures found, 4
1 Discover possible sources
2 Establish search terms
3 Search for pictures
4. Request photocopies of found items
do 3.1, 3.2, 3.3 until pictures found, 3.4 if necessary , 3.5
3.1 Select library
3.2 Log in to catalogue
3.3 Search for pictures
3.4 Modify search terms
3.5 Record relevant items
do 3.3.1, 3.3.2, 3.3.3
3.3.1 Enter search terms
3.3.2 Initiate search
3.3.3 Review results

Interviewing:

- Design semi-structured interviews based on open-ended questions.
- Users can then provide information that they think is essential; not just information that you have thought of collecting.
- Group interviews or focus groups allow users to discuss with each other what they do.

Ethnography:

- Involves an external observer watching users at work and questioning them in an unscripted way about their work.
- Valuable because many user tasks are intuitive and they find these very difficult to describe and explain.
- Also helps understand the role of social and organizational influences on work

User Interface Prototyping:

- The aim of prototyping is to allow users to gain direct experience with the interface.
- Without such direct experience, it is impossible to judge the usability of an interface.
- Prototyping may be a two-stage process:
  - Early in the process, paper prototypes may be used;
  - The design is then refined and increasingly sophisticated automated prototypes are then developed.

Paper Prototyping Techniques:

- Work through scenarios using sketches of the interface.
- Use a storyboard to present a series of interactions with the system.
- Paper prototyping is an effective way of getting user reactions to a design proposal

Prototyping Techniques:

- Script-driven prototyping
  - Develop a set of scripts and screens using a tool such as Macromedia Director. When the user interacts with these, the screen changes to the next display.
- Visual programming
  - Use a language designed for rapid development such as Visual Basic.
- Internet-based prototyping
  - Use a web browser and associated scripts.

User Interface Evaluation:

- Some evaluation of a user interface design should be carried out to assess its suitability.
- Full scale evaluation is very expensive and impractical for most systems.
- Ideally, an interface should be evaluated against a usability specification. However, it is rare for such specifications to be produced.

Simple Evaluation Techniques:

- Questionnaires for user feedback.
- Video recording of system use and subsequent tape evaluation.
- Instrumentation of code to collect information about facility use and user errors.

The provision of code in the software to collect on-line user feedback.

b. What are fault tolerance actions? Explain them briefly (4)

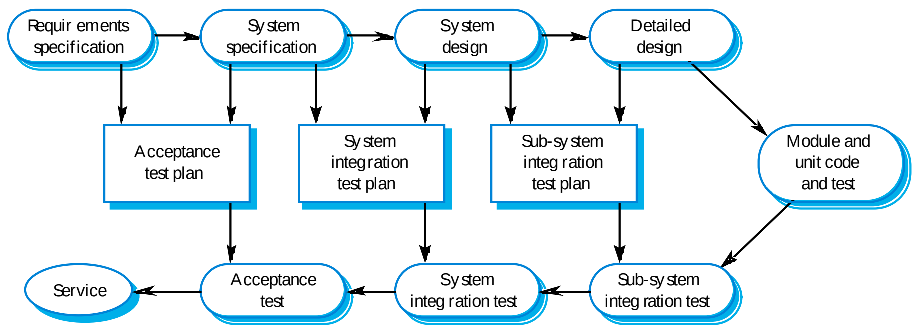
Answer:

- Fault detection - The system must detect that a fault (an incorrect system state) has occurred.
- Damage assessment -The parts of the system state affected by the fault must be detected.
- Fault recovery -The system must restore its state to a known safe state.
- Fault repair -The system may be modified to prevent recurrence of the fault. As many software faults are transitory, this is often unnecessary.

Q.8 a. Explain the planning involved in verification and validation. (5)

Answer:

- Careful planning is required to get the most out of testing and inspection processes.
- Planning should start early in the development process.
- The plan should identify the balance between static verification and testing.
- Test planning is about defining standards for the testing process rather than describing product tests.


Requir ements specification
System specification
System design
Detailed design
Acceptance test plan
System integration test plan
Sub-system integ ration test plan
Module and unit code and test
Service
Acceptance test
System integ ration test
Sub-system integ ration test

b. Explain System Testing. (3)

Answer:

- Involves integrating components to create a system or sub-system.
- May involve testing an increment to be delivered to the customer.
- Two phases:

- Integration testing - the test team have access to the system source code. The system is tested as components are integrated.
- Release testing - the test team test the complete system to be delivered as a black-box

c. Explain COCOMO model briefly **(8)**

Answer:

- An empirical model based on project experience.
- Well-documented, 'independent' model which is not tied to a specific software vendor.
- Long history from initial version published in 1981 (COCOMO-81) through various instantiations to COCOMO 2.
- COCOMO 2 takes into account different approaches to software development, reuse, etc.

Project complexity	**Formula**	**Description**
Simple	$PM = 2.4\ (KDSI)^{1.05} \times M$	Well-understood applications developed by small teams.
Moderate	$PM = 3.0\ (KDSI)^{1.12} \times M$	More complex projects where team members may have limited experience of related systems.
Embedded	$PM = 3.6\ (KDSI)^{1.20} \times M$	Complex projects where the software is part of a strongly coupled complex of hardware, software, regulations and operational procedures.

COCOMO 2 incorporates a range of sub-models that produce increasingly detailed software estimates.

The sub-models in COCOMO 2 are:

Application composition model. Used when software is composed from existing parts.

Early design model. Used when requirements are available but design has not yet started.

Reuse model. Used to compute the effort of integrating reusable components.

Post-architecture model. Used once the system architecture has been designed and more information about the system is available.

Q.9 a. Discuss ISO 9000 Standards for Quality Management. **(8)**

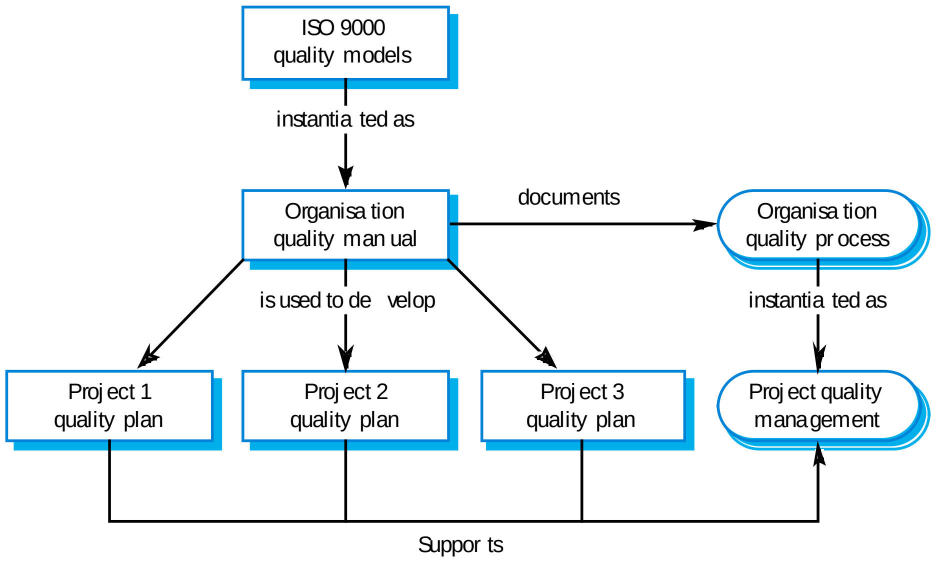
Answer:

- An international set of standards for quality management.
- Applicable to a range of organisations from manufacturing to service industries.
- ISO 9001 applicable to organisations which design, develop and maintain products.
- ISO 9001 is a generic model of the quality process that must be instantiated for each organisation using the standard

Management responsibility	**Quality system**
Control of non-conforming products	Design control
Handling, storage, packaging and delivery	Purchasing
Purchaser-supplied products	Product identification and traceability
Process control	Inspection and testing
Inspection and test equipment	Inspection and test status
Contract review	Corrective action
Document control	Quality records
Internal quality audits	Training
Servicing	Statistical techniques

Documentation Standards:

- o Document identification standards - How documents are uniquely identified.



- o Document structure standards - Standard structure for project documents.
- o Document presentation standards - Define fonts and styles, use of logos, etc.

- Document update standards
- Define how changes from previous versions are reflected in a document

Documentation Interchange Standards:

- Interchange standards allow electronic documents to be exchanged, mailed, etc.
- Documents are produced using different systems and on different computers. Even when standard tools are used, standards are needed to define conventions for their use e.g. use of style sheets and macros.
- Need for archiving. The lifetime of word processing systems may be much less than the lifetime of the software being documented. An archiving standard may be defined to ensure that the document can be accessed in future.

b. What are the attributes of Process improvement attributes. **(4)**

Answer:

Process characteristic	**Description**
Understandability	To what extent is the process explicitly defined and how easy is it to understand the process definition?
Visibility	Do the process activities culminate in clear results so that the progress of the process is externally visible?
Supportability	To what extent can CASE tools be used to support the process activities?
Acceptability	Is the defined process acceptable to and usable by the engineers responsible for producing the software product?
Reliability	Is the process designed in such a way that process errors are avoided or trapped before they result in product errors?
Robustness	Can the process continue in spite of unexpected problems?
Maintainability	Can the process evolve to reflect changing organisational requirements or identified process improvements?
Rapidity	How fast can the process of delivering a system from a given specification be completed?

c. Write short notes on Configuration Management Planning. **(4)**

Answer:

- All products of the software process may have to be managed:
  - Specifications;
  - Designs;
  - Programs;
  - Test data;
  - User manuals.
- Thousands of separate documents may be generated for a large, complex software system.
- Defines the types of documents to be managed and a document naming scheme.
- Defines who takes responsibility for the CM procedures and creation of baselines.
- Defines policies for change control and version management.
- Defines the CM records which must be maintained.
- Describes the tools which should be used to assist the CM process and any limitations on their use.
- Defines the process of tool use.
- Defines the CM database used to record configuration information.
- May include information such as the CM of external software, process auditing, etc.