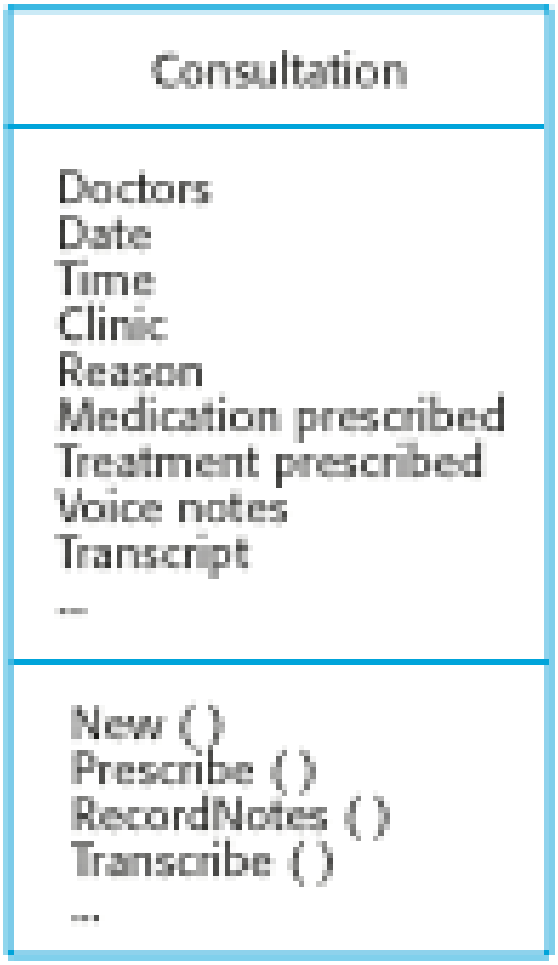
Answer Key- II Internal

Q1 Explain in detail the structural models of a software system.
Ans Structural models of software display the organization of a system in terms of the components that make up that system and their relationships. Structural models may be static models, which show the structure of the system design, or dynamic models, which show the organization of the system when it is executing. We create structural models of a system when you are discussing and designing the system architecture.

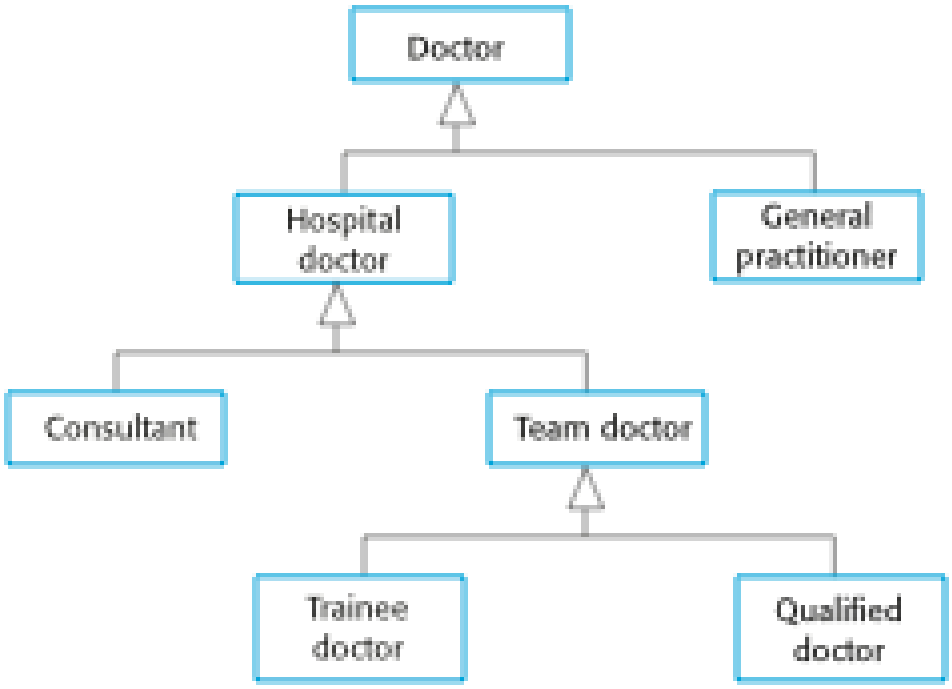
Class diagrams

- ✧ Class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes. An object class can be thought of as a general definition of one kind of system object. An association is a link between classes that indicates that there is some relationship between these classes. When you are developing models during the early stages of the software engineering process, objects represent something in the real world, such as a patient, a prescription, doctor, etc.



Generalization

- ✧ Generalization is an everyday technique that we use to manage complexity. Rather than learn the detailed characteristics of every entity that we experience, we place these entities in more general classes (animals, cars, houses, etc.) and learn the characteristics of these classes. This allows us to infer that different members of these classes have some common characteristics e.g. squirrels and rats are rodents.



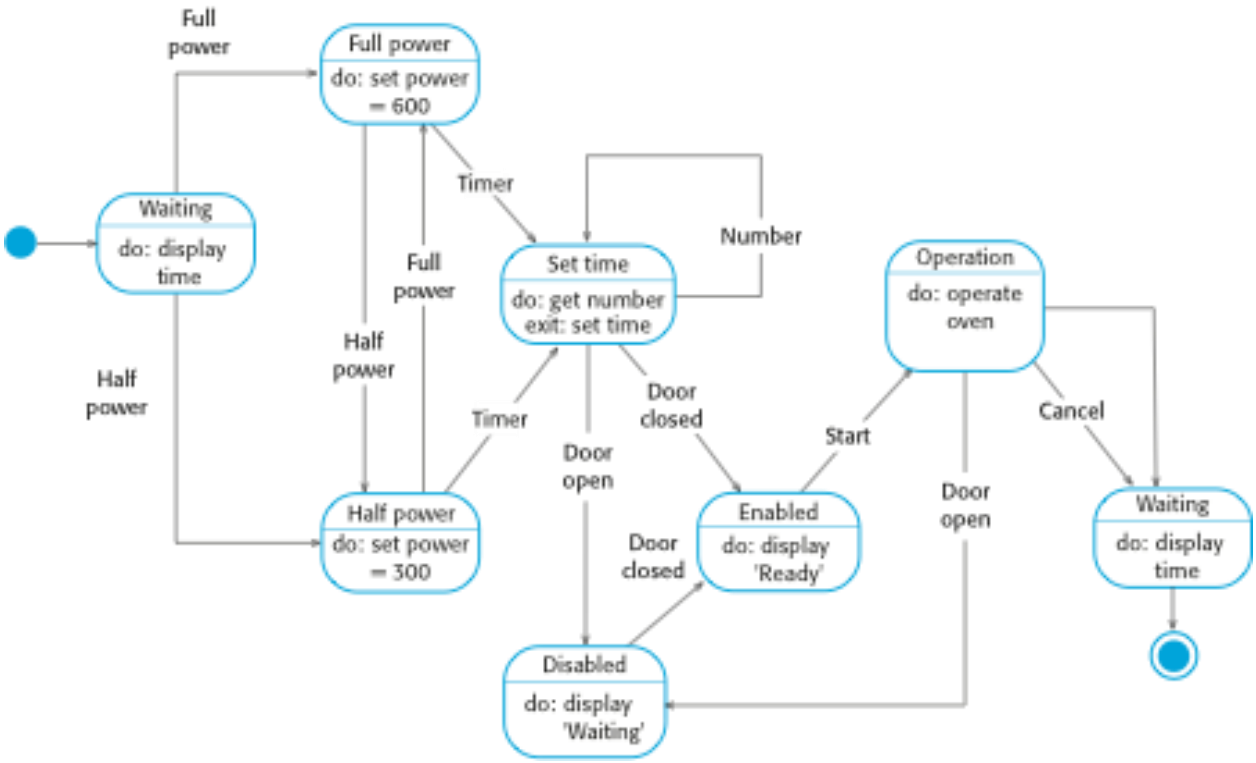
Q2 Explain event Driven and data driven modeling.

- ✧ Ans Real-time systems are often event-driven, with minimal data processing. For example, a landline phone switching system responds to events such as ‘receiver off hook’ bygenerating a dial tone.
- ✧ Event-driven modeling shows how a system responds to external and internal events.
- ✧ It is based on the assumption that a system has a finite number of states and that events (stimuli) may cause a transition from one state to another.

State machine models

- ✧ These model the behaviour of the system in response to external and internal events.
- ✧ They show the system’s responses to stimuli so are often used for modelling real-time systems.
- ✧ State machine models show system states as nodes and events as arcs between these nodes. When an event occurs, the system moves from one state to another.
- ✧ Statecharts are an integral part of the UML and are used to represent state machine models.

State diagram of a microwave oven



States and stimuli for the microwave oven

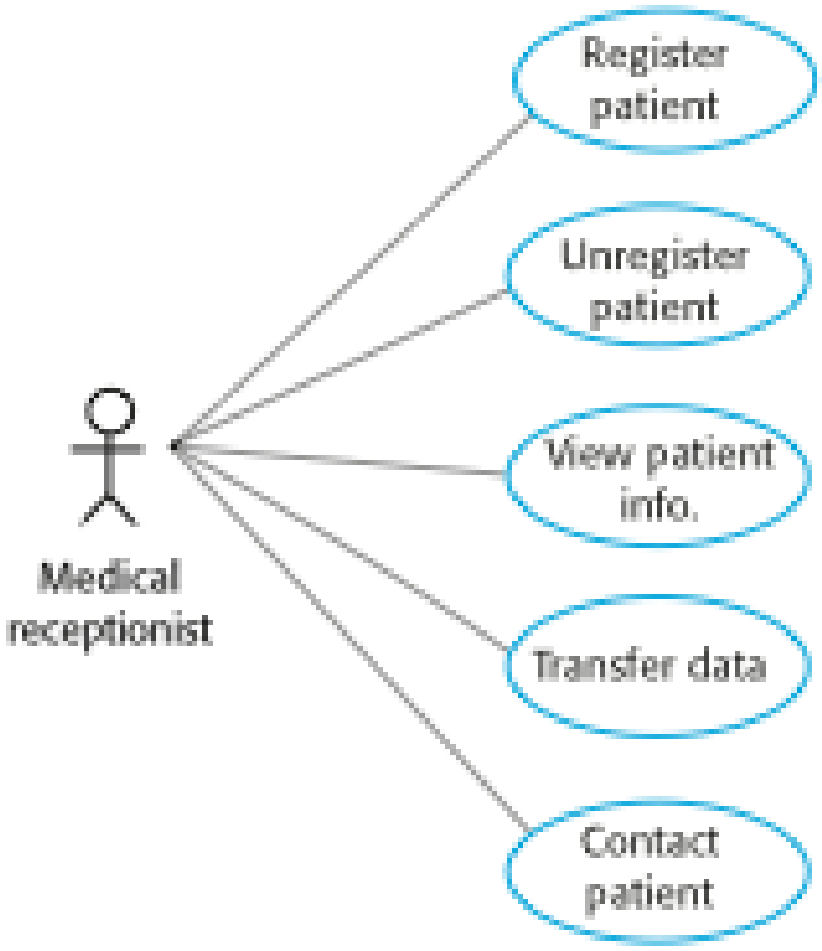
State	Description
Waiting	The oven is waiting for input. The display shows the current time.
Half power	The oven power is set to 300 watts. The display shows ‘Half power’.
Full power	The oven power is set to 600 watts. The display shows ‘Full power’.
Set time	The cooking time is set to the user’s input value. The display shows the cooking time selected and is updated as the time is set.
Disabled	Oven operation is disabled for safety. Interior oven light is on. Display shows ‘Not ready’.
Enabled	Oven operation is enabled. Interior oven light is off. Display shows ‘Ready to cook’.

Operation	Oven in operation. Interior oven light is on. Display shows the timer countdown. On completion of cooking, the buzzer is sounded for five seconds. Oven light is on. Display shows 'Cooking complete' while buzzer is sounding.

Q3 Explain Use case diagram and sequence diagram with the help of an example.

- ✧ Ans Use cases were developed originally to support requirements elicitation and now incorporated into the UML. Each use case represents a discrete task that involves external interaction with a system. Actors in a use case may be people or other systems. Represented diagrammatically to provide an overview of the use case and in a more detailed textual form.
- ✧ A use case in the MHC-PMS

MHC-PMS: Transfer data	
Actors	Medical receptionist, patient records system (PRS)
Description	A receptionist may transfer data from the MHC-PMS to a general patient record database that is maintained by a health authority. The information transferred may either be updated personal information (address, phone number, etc.) or a summary of the patient's diagnosis and treatment.
Data	Patient's personal information, treatment summary
Stimulus	User command issued by medical receptionist
Response	Confirmation that PRS has been updated
Comments	The receptionist must have appropriate security permissions to access the patient information and the PRS.

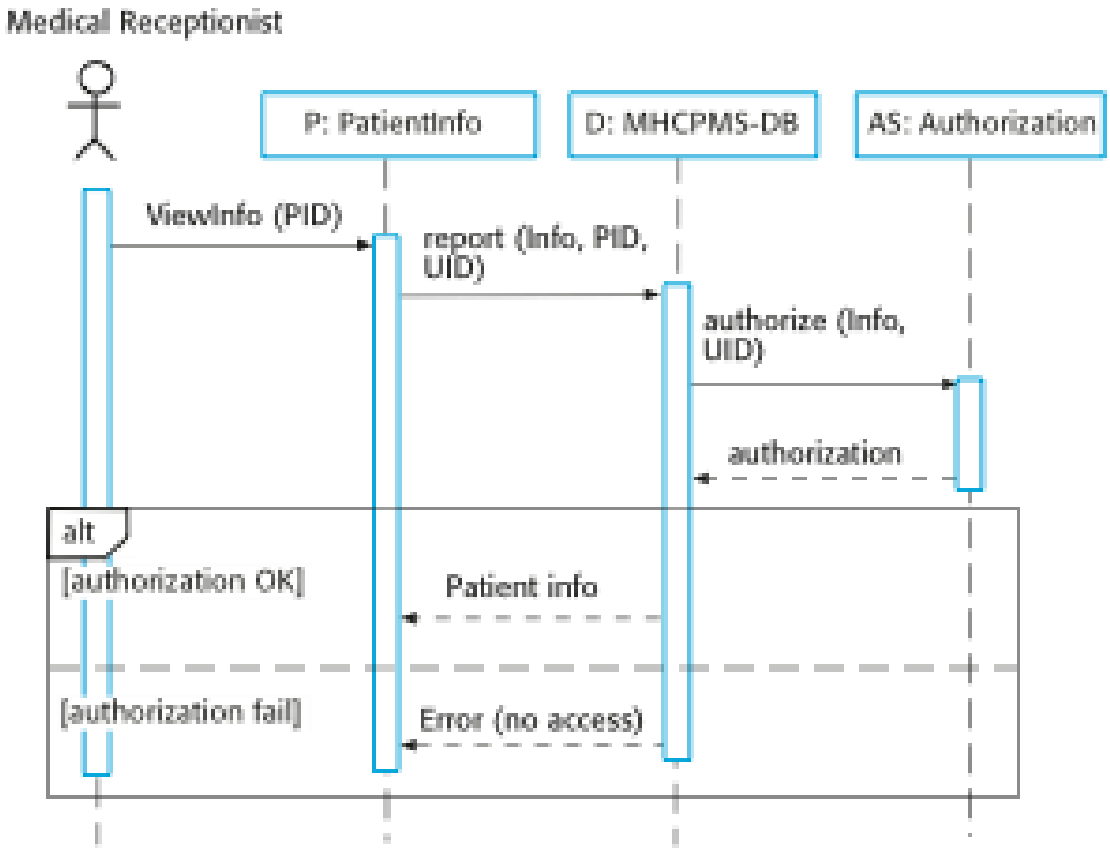


Use cases in the MHC-PMS involving the role 'Medical Receptionist'

Sequence diagrams

- Sequence diagrams are part of the UML and are used to model the interactions between the actors and the objects within a system. A sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance. The objects and actors involved are listed along the top of the diagram, with a dotted line drawn vertically from these. Interactions between objects are indicated by annotated arrows.

Sequence diagram for View patient information

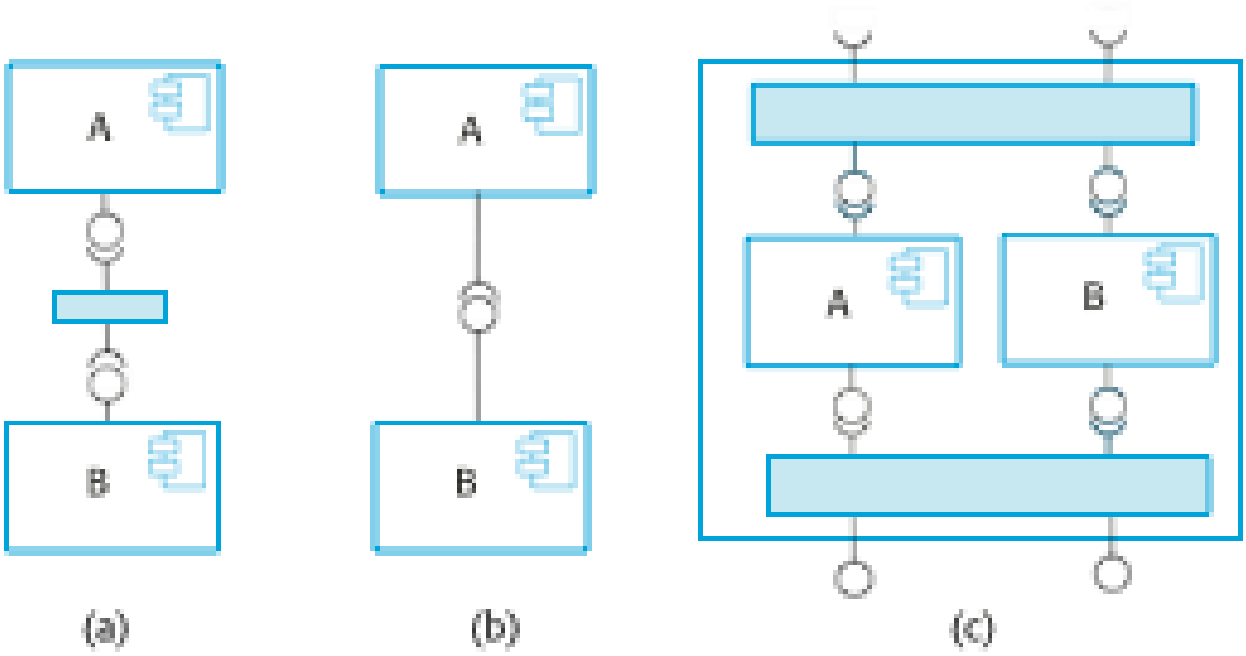


Q4 Explain Component Composition in detail.

Ans The process of assembling components to create a system is called component composition. Composition involves integrating components with each other and with the component infrastructure. Normally we have to write ‘glue code’ to integrate components.

Types of composition

- ✧ Sequential composition where the composed components are executed in sequence. This involves composing the provides interfaces of each component.
- ✧ Hierarchical composition where one component calls on the services of another. The provides interface of one component is composed with the requires interface of another.
- ✧ Additive composition where the interfaces of two components are put together to create a new component. Provides and requires interfaces of integrated component is a combination of interfaces of constituent components.



✧ **Interface incompatibility**

- ✧ Parameter incompatibility where operations have the same name but are of different types.
- ✧ Operation incompatibility where the names of operations in the composed interfaces are different.
- ✧ Operation incompleteness where the provides interface of one component is a subset of the requires interface of another.

Adaptor components

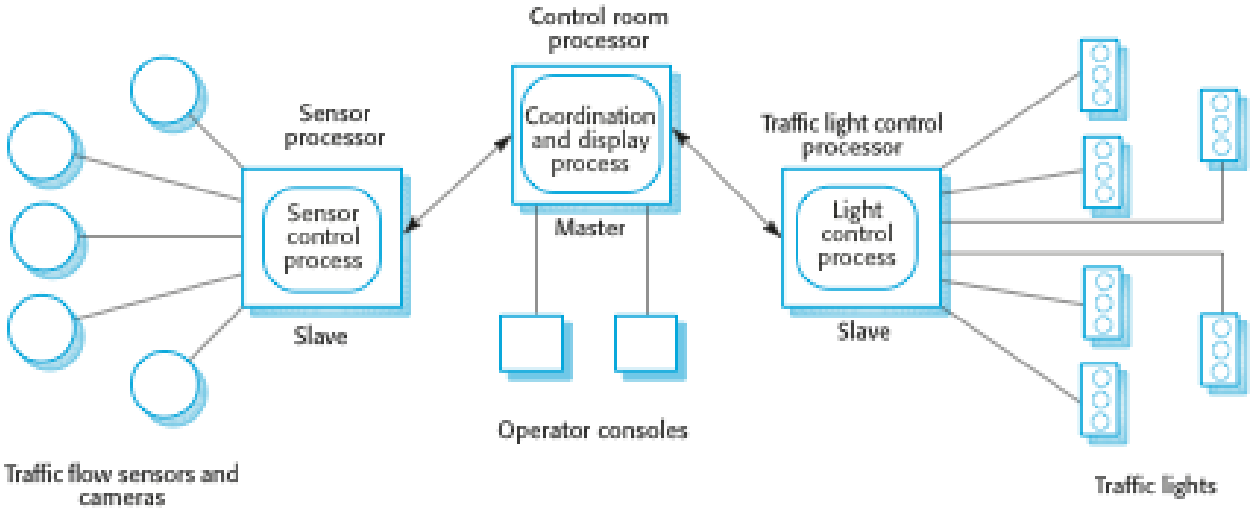
- ✧ Address the problem of component incompatibility by reconciling the interfaces of the components that are composed. Different types of adaptor are required depending on the type of composition.

Q5 Explain Architectural patterns for distributed systems.

Ans Widely used ways of organizing the architecture of a distributed system:

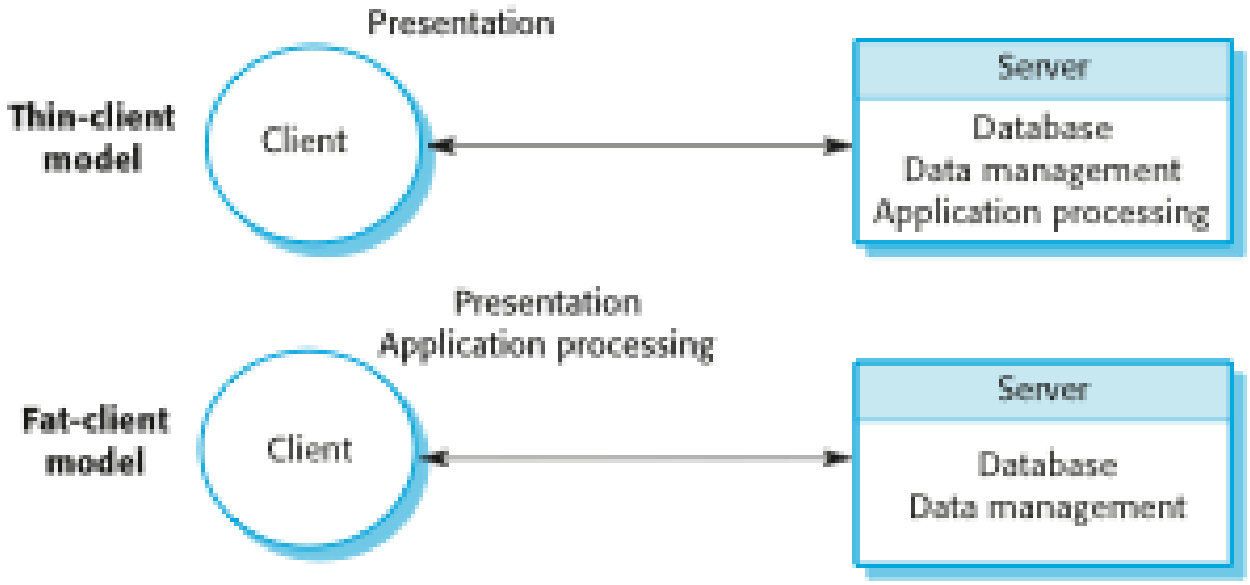
- Master-slave architecture, which is used in real-time systems in which guaranteed interaction response times are required. Master-slave architectures are commonly used in real-time systems where there may be separate processors associated with data acquisition from the

system's environment, data processing and computation and actuator management. The 'master' process is usually responsible for computation, coordination and communications and it controls the 'slave' processes. 'Slave' processes are dedicated to specific actions, such as the acquisition of data from an array of sensors.

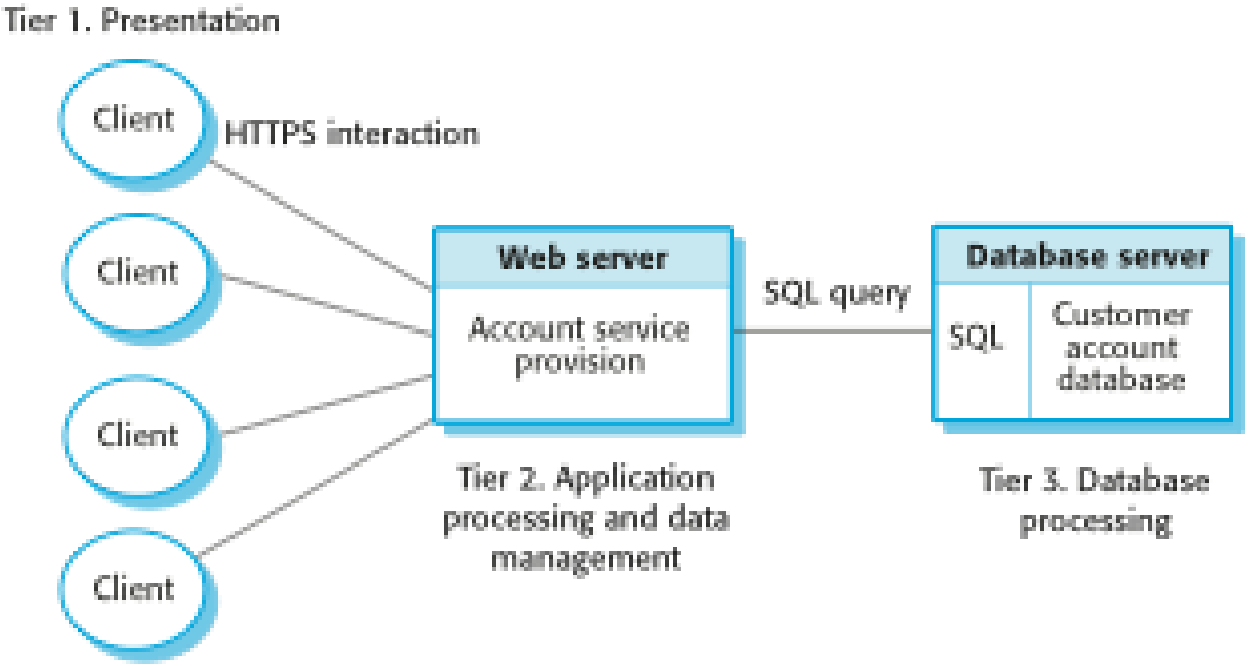


-

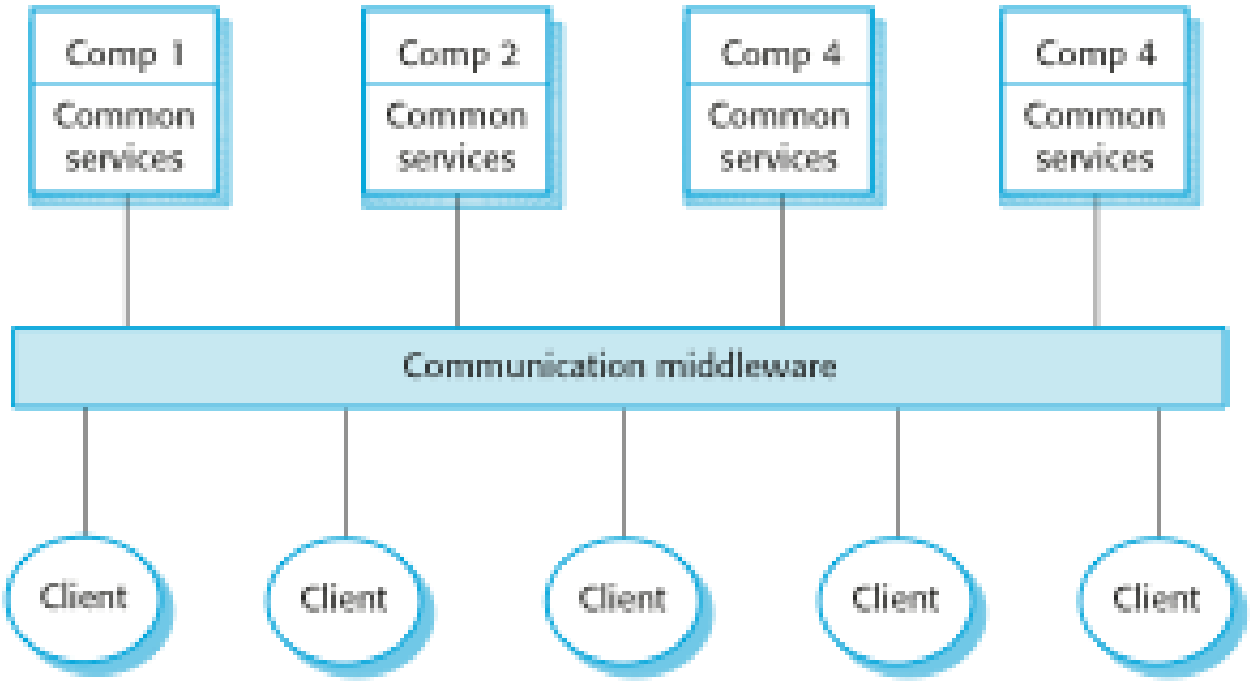
  - Two-tier client-server architecture, which is used for simple client-server systems, and where the system is centralized for security reasons.In a two-tier client-server architecture, the system is implemented as a single logical server plus an indefinite number of clients that use that server.
  - Thin-client model, where the presentation layer is implemented on the client and all other layers (data management, application processing and database) are implemented on a server.
  - Fat-client model, where some or all of the application processing is carried out on the client. Data management and database functions are implemented on the server.



- Multi-tier client-server architecture, which is used when there is a high volume of transactions to be processed by the server. In a 'multi-tier client–server' architecture, the different layers of the system, namely presentation, data management, application processing, and database, are separate processes that may execute on different processors. This avoids problems with scalability and performance if a thin-client two-tier model is chosen, or problems of system management if a fat-client model is used.

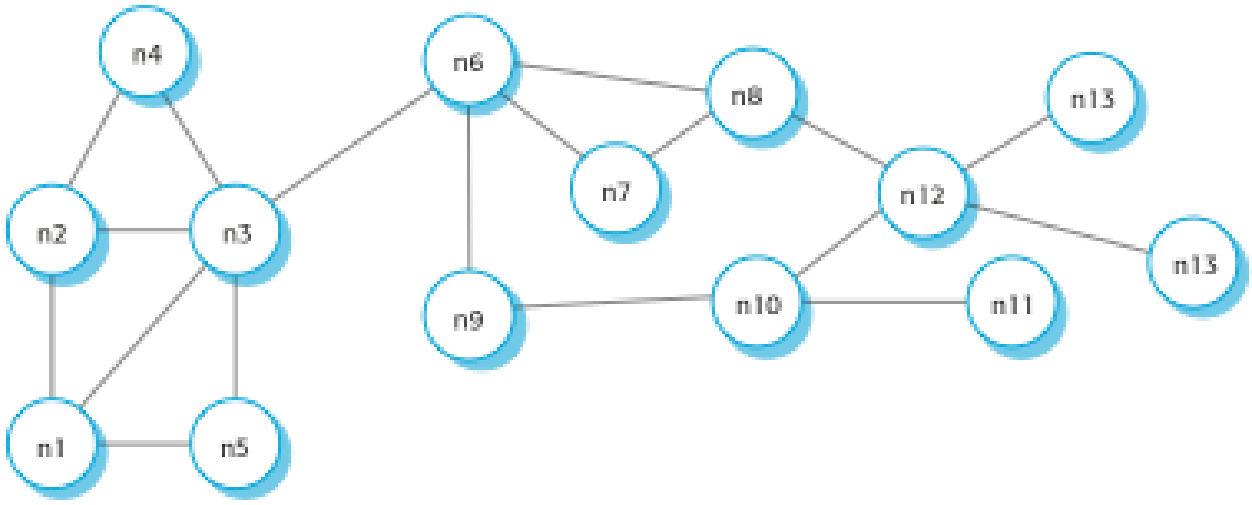


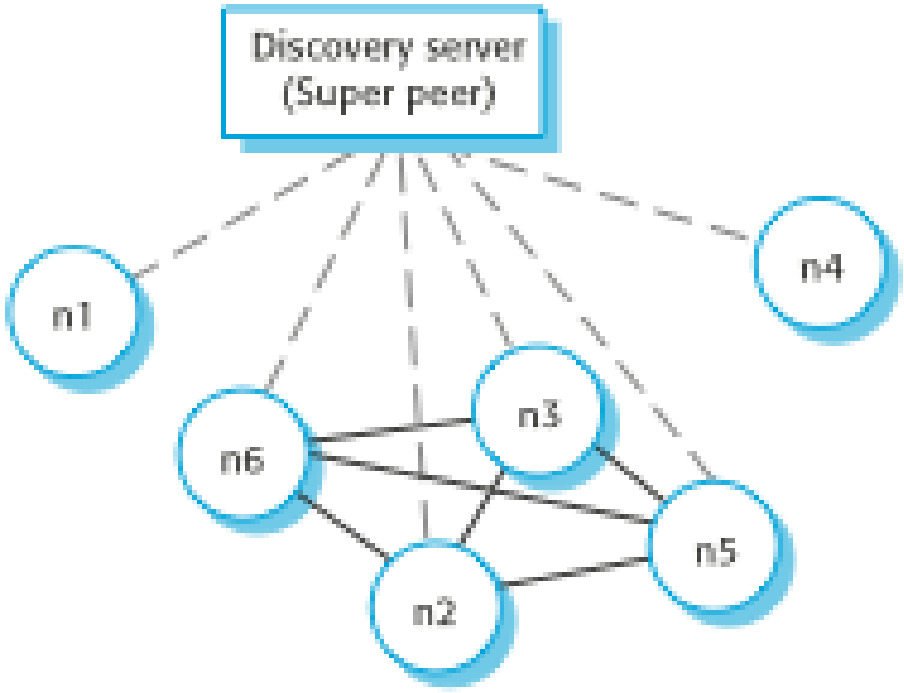
- ✧ Distributed component architecture, which is used when resources from different systems and databases need to be combined, or as an implementation model for multi-tier client-server systems. There is no distinction in a distributed component architectures between clients and servers. Each distributable entity is an object that provides services to other components and receives services from other components. Component communication is through a middleware system. However, distributed component architectures are more complex to design than C/S systems.



  - Peer-to-peer architecture, which is used when clients exchange locally stored information and the role of the server is to introduce clients to each other. Peer to peer (p2p) systems are decentralised systems where computations may be carried out by any node in the network. The overall system is designed to take advantage of the computational power and storage of a large number of networked computers. Most p2p systems have been personal systems but there is increasing business use of this technology.

- **A decentralized p2p architecture**





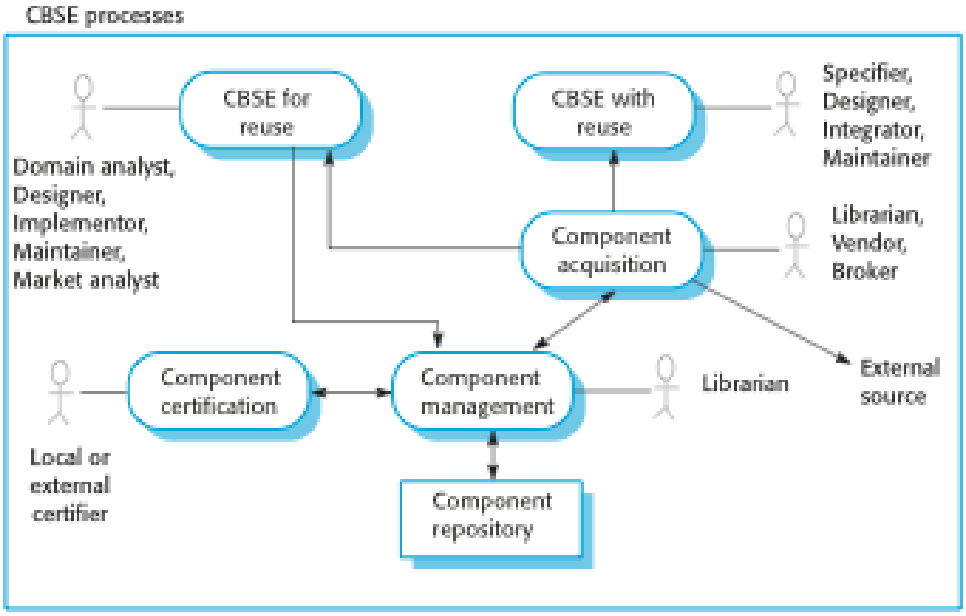
A semicentralized p2p architecture

Q6 What are CBSE processes? Explain basic process of CBSE model.

- Ans CBSE processes are software processes that support component-based software engineering. They take into account the possibilities of reuse and the

different process activities involved in developing and using reusable components.

- ✧ Development for reuse
  - This process is concerned with developing components or services that will be reused in other applications. It usually involves generalizing existing components.
- ✧ Development with reuse
  - This process is the process of developing new applications using existing components and services.



- Component acquisition is the process of acquiring components for reuse or development into a reusable component. It may involve accessing locally-developed components or services or finding these components from an external source.

- ✧ Component management is concerned with managing a company's reusable components, ensuring that they are properly catalogued, stored and made available for reuse.
- ✧ Component certification is the process of checking a component and certifying that it meets its specification.
- ✧ CBSE for reuse focuses on component development. Components developed for a specific application usually have to be generalised to make them reusable. A component is most likely to be reusable if it associated with a stable domain abstraction (business object). For example, in a hospital stable domain abstractions are associated with the fundamental purpose - nurses, patients, treatments, etc.

- ✧ CBSE with reuse process has to find and integrate reusable components. When reusing components, it is essential to make trade-offs between ideal requirements and the services actually provided by available components.
- ✧ This involves:
  - Developing outline requirements;

- Searching for components then modifying requirements according to available functionality.
- Searching again to find if there are better components that meet the revised requirements.
- Composing components to create the system.

Q7 Explain about any one White Box testing method and any one Black Box testing method.

Ans There are two basic approaches to designing the test cases to be used in testing: black-box and white-box.

Black-Box Testing

In black-box testing the structure of the program is not considered. Test cases are decided solely on the basis of the requirements or specifications of the program or module, and the internals of the module or the program are not considered for selection of test cases. In black-box testing, the tester only knows the inputs that can be given to the system and what output the system should give.

Boundary Value Analysis

This is the one of the approach for black box testing. It has been observed that programs that work correctly for a set of values in an equivalence class fail on some special values. These values often lie on the boundary of the equivalence class. Test cases that have values on the boundaries of equivalence classes are therefore likely to be "high-yield" test cases, and selecting such test cases is the aim of boundary value analysis. In boundary value analysis , we choose an input for a test case from an equivalence class, such that the input lies at the edge of the equivalence classes. Boundary values for each equivalence class, including the equivalence classes of the output, should be covered. Boundary value test cases are also called "extreme cases." Hence, we can say that a boundary value test case is a set of input data that lies on the edge or boundary of a class of input data or that generates output that lies at the boundary of a class of output data. In case of ranges, for boundary value analysis it is useful to select the boundary elements of the range and an invalid value just beyond the two ends (for the two invalid equivalence classes). So, if the range is 0.0 _ x _ 1.0, then the test cases are 0.0, 1.0 (valid inputs), and −0.1, and 1.1 (for invalid inputs). Similarly, if the input is a list, attention should be focused on the first and last elements of the list.

White-Box Testing

White-box testing, on the other hand, is concerned with testing the implementation of the program. The intent of this testing is not to exercise all the different input or output conditions (although that may be a by-product) but to exercise the different programming structures and data structures used in the program. White-box testing is also called structural testing, and we will use the two terms interchangeably. Here we will discuss one approach to structural testing: control flow-based testing, which is most commonly used in practice.

Control Flow-Based Criteria

Most common structure-based criteria are based on the control flow of the program. In these criteria, the control flow graph of a program is considered and coverage of various aspects of the graph are specified as criteria. Perhaps the simplest coverage criterion is statement coverage, which requires that each statement of the program be executed at least once during testing. In other words, it requires that the paths executed during testing include all the nodes in the graph. This is also called the all-nodes criterion .This coverage criterion is not very strong, and can

leave errors undetected. For example, if there is an if statement in the program without having an else clause, the statement coverage criterion for this statement will be satisfied by a test case that evaluates the condition to true. No test case is needed that ensures that the condition in the if statement evaluates to false. A more general coverage criterion is branch coverage, which requires that each edge in the control flow graph be traversed at least once during testing.
In other words, branch coverage requires that each decision in the program be evaluated to true and false values at least once during testing. The trouble with branch coverage comes if a decision has many conditions in it Hence, a more general coverage criterion is one that requires all possible paths in the control flow graph be executed during testing. This is called the path coverage criterion or the all-paths criterion, and the testing based on this criterion is often called path testing.

Q8 Describe in detail the process of testing.

Testing Process

The testing process for a project consists of three high-level tasks—test planning, test case design, and test execution.

Test Plan

A test plan is a general document for the entire project that defines the scope, approach to be taken, and the schedule of testing, as well as identifies the test items for testing and the personnel responsible for the different activities of testing. The test planning can be done well before the actual testing commences and can be done in parallel with the coding and design activities. The inputs for forming the test plan are: (1) project plan, (2) requirements document, and (3) architecture or design document. The project plan is needed to make sure that the test plan is consistent with the overall quality plan for the project and the testing schedule matches that of the project plan. The requirements document and the design document are the basic documents used for selecting the test units and deciding the approaches to be used during testing. A test plan should contain the following:

– Test unit specification
– Features to be tested
– Approach for testing
– Test deliverables
– Schedule and task allocation

Test Case Design

Test case design has to be done separately for each unit. Based on the approach specified in the test plan, and the features to be tested, the test cases are designed and specified for testing the unit. Test case specification gives, for each unit to be tested, all test cases, inputs to be used in the test cases, conditions being tested by the test case, and outputs expected for those test cases. It is therefore important to ensure that the set of test cases used is of high quality. Evaluation of test cases is often done through test case review.

Test Case Execution

With the specification of test cases, the next step in the testing process is to execute them. This step is also not straightforward. The test case specifications only specify the set of test cases for the unit to be tested. However, executing the test cases may require construction of driver modules or stubs. It may also require modules to set up the environment as stated in the test plan

and test case specifications. Only after all these are ready can the test cases be executed. During test case execution, defects are found. These defects are then fixed and testing is done again to verify the fix. To facilitate reporting and tracking of defects found during testing (and other quality control activities), defects found are often logged. Defect logging is particularly important in a large software project which may have hundreds or thousands of defects that are found by different people at different stages of the project. Often the person who fixes a defect is not the person who finds or reports the defect.

A defect can be found by anyone at anytime. When a defect is found, it is logged in a defect control system, along with sufficient information about the defect. The defect is then in the state "submitted," essentially implying that it has been logged along with information about it. The job of fixing the defect is then assigned to some person, who is generally the author of the document or code in which the defect is found. The assigned person does the debugging and fixes the reported defect, and the defect then enters the "fixed" state. However, a defect that is fixed is still not considered as fully done. The successful fixing of the defect is verified. This verification may be done by another person (often the submitter), or by a test team, and typically involves running some tests. Once the defect fixing is verified, then the defect can be marked as "closed." In other words, the general life cycle of a defect has three states—submitted, fixed, and closed, as shown in figure. A defect that is not closed is also called open.