EX NO: **DATE:**	**To Identify a software system that needs to be developed – Software Personnel Management System**

Aim:

To identify the need for a Software Personnel Management System and provide an overview of its features and architecture.

Introduction:

In the realm of modern business operations, efficient management of personnel is paramount for organizational success. A Software Personnel Management System serves as a comprehensive solution for managing various aspects of personnel administration, ensuring streamlined operations and maximizing productivity. This introduction highlights the importance of such a system and outlines its key functionalities.

Features of Software Personnel Management System:

1. Employee Database Management:

- Allow administrators to manage employee profiles, including personal details, job roles, and performance evaluations.
- Implement authentication and authorization mechanisms to ensure secure access to employee information.
- Define different employee categories and access levels, such as regular employees, managers, and HR personnel.

2. Attendance and Leave Management:

- Facilitate recording and monitoring of employee attendance, including clock-in/clock-out times and leave requests.
- Provide tools for employees to request various types of leave (e.g., sick leave, vacation) and for managers to approve or reject these requests.
- Generate reports on employee attendance patterns and leave utilization for HR analysis.

3. Performance Evaluation and Feedback:

- Incorporate features for conducting performance evaluations, setting goals, and tracking progress.
- Enable managers to provide regular feedback and performance reviews to employees.
- Implement mechanisms for employees to submit self-assessments and development plans.

4. Training and Development:

- Offer modules for managing employee training programs, including course registration, tracking completion, and assessing effectiveness.
- Provide resources for employees to access training materials and resources relevant to their roles.
- Generate reports on training progress and skills development across the organization.

5. Payroll and Benefits Administration:

- Integrate payroll processing functionality to calculate salaries, deductions, and benefits for employees.
- Manage employee benefits packages, including health insurance, retirement plans, and other perks.
- Ensure compliance with tax regulations and labor laws governing payroll and benefits administration.

Architecture of Software Personnel Management System:

6. Client-Server Architecture:

- Adopt a client-server architecture where the web application serves as the client, and the server hosts the application logic and data.
- Utilize RESTful APIs for communication between the client interface and server-side components.

7. Relational Database Schema:

- Design a relational database schema to store employee information, attendance records, performance evaluations, and payroll data.
- Ensure data integrity, normalization, and efficient querying for optimal system performance.

Result:

Thus the software system that is need to be developed for Software Personnel Management System was executed successfully.

Ex No: **Date:**	**DOCUMENT THE SOFTWARE REQUIREMENTS SPECIFICATION(SRS) FOR THE IDENTIFIED SYSTEM**

Aim:

To analyze, Design and develop code for Software Personnel Management system using Rational Rose software and Visual Basic 6.0.

Problem statement:

Software personnel system is a real time application used in the developer's day to day system. This is a database to store the project configuration that is specified by the customer and the software developed for it includes customer and developer requirements needed for the process. Here we assume ourselves as the developing company:

1. The company is the developer of the software and it contains the necessary information of the software such as requirements, time limit etc.
2. The customer provides the requirements to the company and requests a suitable solution system. The company is only responsible for developing the system out of the requirements of the customer.
3. The company mainly obtains the requirements from the customer and by analyzing it, prepares a requirements and feasibility documentation initially.
4. Then the company develops a software system that is apt for the integrated requirements and tests it within itself and by exposure to the concerned customer, and finally builds a complete a satisfactory solution.

1. Introduction

1.1 Purpose

The Software Personnel Management System (SPMS) is designed to streamline and automate personnel-related tasks within a software development organization. It aims to enhance efficiency and organization by providing comprehensive tools for managing employee data, scheduling, performance evaluation, and communication.

1.2 Document Conventions

	Font	**Style**	**Size**
Heading	Times New Roman	Bold	18
Sub-Heading	Times New Roman	Bold	14
Other's	Times New Roman	Regular	11

1.3 Intended Audience and Reading Suggestions

This document is intended for the project development team, including project managers, developers, testers, and documentation writers. It serves as a reference for stakeholders involved in personnel management.

1.4 Product Scope

The SPMS encompasses features for employee information management, task scheduling, performance evaluation, and communication. It aims to improve productivity and organization within the software development team.

1.5 Definitions, Acronyms and Abbreviations

- SPMS: Software Personnel Management System
- HR: Human Resources
- GUI: Graphical User Interface
- API: Application Programming Interface

2. Overall Description

2.1 Product Perspective

The SPMS acts as an integrated platform for managing personnel-related tasks within the software development organization. It interfaces with HR systems, databases, and communication tools to provide a comprehensive solution.

2.2 Product Functions

- Employee Information Management
- Task Scheduling and Assignment
- Performance Evaluation and Feedback
- Communication and Collaboration

2.3 User Classes and Characteristics

Administrators: HR personnel and managers responsible for system administration.

Employees: Software developers, testers, and other team members who interact with the system.

2.4 Operating Environment

Client System:

- Operating System: Windows/Linux/macOS
- Processor: Intel or AMD
- RAM: 4GB or higher

Server System:

- Operating System: Linux
- Processor: Intel or AMD
- RAM: 8GB or higher

2.5 Design and Implementation Constraints

- Accessibility via web browsers
- Implementation of security measures to protect sensitive data
- Integration with existing HR systems may require API access

2.6 Assumptions and Dependencies

- Basic computer literacy among users
- Reliable internet connectivity
- Access control based on user roles and permissions

3. External Interface Requirements

3.1 User Interfaces

Intuitive GUIs for administrators and employees to perform various tasks such as data entry, scheduling, and performance evaluation.

3.2 Hardware Interfaces

Standard hardware configurations for client systems accessing the system via web browsers.

3.3 Software Interfaces

- Front End: HTML, CSS, JavaScript
- Back End: Java, Spring Framework
- Database: MySQL or similar RDBMS

3.4 Communications Interfaces

Integration with external APIs for HR systems and other tools.

4. System Functions

- Employee Information Management
- Task Scheduling and Assignment
- Performance Evaluation and Feedback
- Communication and Collaboration

5. Other Nonfunctional Requirements

5.1 Performance Requirements

System response time within 2 seconds under normal load conditions.

5.2 Safety Requirements

Access control for sensitive data based on user roles and permissions.

5.3 Security Requirements

Implementation of data encryption and secure authentication mechanisms.

5.4 Software Quality Attributes

Emphasis on usability, reliability, and scalability.

5.5 Business Rules

Enforcement of principles such as confidentiality, data integrity, and compliance with labor laws.

Appendix A: Glossary

- SPMS: Software Personnel Management System
- HR: Human Resources
- GUI: Graphical User Interface
- API: Application Programming Interface

RESULT:

Thus the project to develop software personal management system is done successfully.

EX NO: DATE:	**To Identify the Use Cases and Develop the Use Case Model**

AIM:

To identify the use cases and develop a use case model for a Software Personnel Management System.

Introduction of Use Case Diagram:

A use case diagram is a graphical representation within the Unified Modeling Language (UML) that illustrates the interactions between a system and its actors, highlighting the system's behavior. It provides a visual overview of the functionalities offered by a system and the roles involved in those functionalities. Use case diagrams are fundamental in software engineering for capturing system requirements and visualizing the system's high-level structure.

Purpose of Use Case Diagram:

The purpose of a use case diagram is to visually depict the functional requirements of a system from the perspective of its users or external systems. It aids in understanding the system's behavior by demonstrating how users interact with the system to accomplish specific tasks or objectives.

Components of Use Case Diagram:

Actors: Actors represent the users or external systems interacting with the system being modeled. In the context of a Software Personnel Management System, actors may include employees, managers, and HR personnel.

Use Cases: Use cases portray the functionalities or services provided by the system. Each use case describes a specific interaction between an actor and the system to achieve a particular goal. For a Software Personnel Management System, examples of use cases include "Recruit," "Training," "Salary," "Monitor," "Performance," "Increment," "Time Management," and "Motivation."

Relationships: Relationships, such as associations and dependencies, depict the connections between actors and use cases. Associations illustrate the interaction between actors and use cases, while dependencies indicate the reliance of one use case on another.

Benefits of Use Case Diagram:

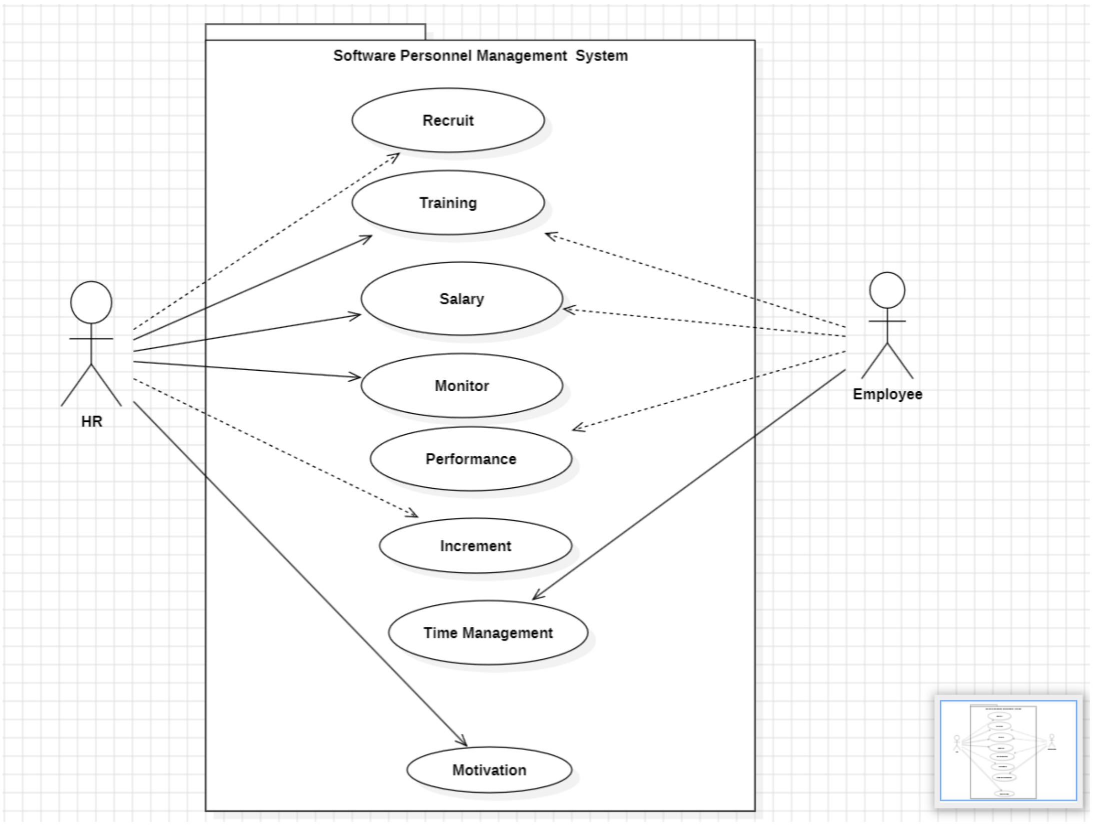
Requirements Elicitation: Use case diagrams aid in eliciting and defining the functional requirements of a system by identifying user interactions and system responses.

Visualization: Use case diagrams provide a visual representation of the system's functionality, facilitating better understanding and analysis of the system's behavior.

Communication: Use case diagrams serve as a communication tool between stakeholders, fostering a shared understanding of the system's requirements and behavior.

Requirements Verification: Use case diagrams enable verification of whether all necessary functionalities and interactions have been captured and addressed in the system design.

Use Cases of Software Personnel Management System:



Recruit Employees:

- HR personnel can create job postings, manage applications, schedule interviews, and hire candidates.
- Managers can review job requirements, interview candidates, and provide feedback to HR personnel.

Training Management:

- HR personnel can schedule training programs, enroll employees, track attendance, and evaluate training effectiveness.
- Employees can view available training programs, register for sessions, complete courses, and provide feedback.

Salary Management:

- HR personnel can manage employee salaries, benefits, allowances, and deductions.
- Employees can view their salary details, access pay stubs, and request adjustments or corrections.

Monitor Performance:

- Managers can set performance goals, monitor employee progress, provide feedback, and conduct performance reviews.
- HR personnel can analyze performance data, identify trends, and generate performance reports.

Increment Processing:

- HR personnel can assess employee performance, recommend salary increments, and update employee compensation.
- Managers can review increment recommendations, provide input, and approve or reject proposed salary adjustments.

Time Management:

- Employees can log their work hours, breaks, and leave requests.

- Managers can review employee timesheets, approve time-off requests, and track attendance.

Motivation Programs:

- HR personnel can design and implement motivation programs, rewards, and recognition initiatives.
- Managers can nominate employees for awards, incentives, or bonuses based on performance and contributions.

Result:

Thus the use case diagram for software personnel management system is implemented and executed successfully.

EX NO: **DATE:**	**Identify the Conceptual classes and Develop a Domain model and Derive Class diagram for Software Personnel management system.**

AIM:

To identify the conceptual classes and develop a domain model and derive class diagram for Software Personnel management system.

Conceptual Classes:

1. Employee: Represents the personnel working within the organization. Attributes may include employee ID, name, position, contact information, etc.

2. Attendance: Represents the attendance records of employees. Attributes may include attendance ID, employee ID, date, time in/out, etc.

3. Leave Request: Represents the requests made by employees for time off. Attributes may include request ID, employee ID, leave type, start/end date, etc.

4. Payroll: Represents the process of managing employee salaries, deductions, and benefits. Attributes may include payroll ID, employee ID, salary, deductions, benefits, etc.

5. HR Manager: Represents the Human Resources manager responsible for overseeing HR-related processes. Attributes may include manager ID, name, contact information, etc.

Domain Model for Software Personnel Management System:

Employee:

Attributes:

- EmployeeID: Integer
- Name: String
- Position: String
- ContactInfo: String

Operations:

- addEmployee()

- deleteEmployee()
- updateEmployee()

Attendance:

Attributes:

- AttendanceID: Integer
- EmployeeID: Integer
- Date: Date
- TimeIn: Time
- TimeOut: Time

Operations:

- markAttendance()

Leave Request:

Attributes:

- RequestID: Integer
- EmployeeID: Integer
- LeaveType: String
- StartDate: Date
- EndDate: Date

Operations:

- submitLeaveRequest()
- approveLeaveRequest()
- rejectLeaveRequest()

Payroll:

Attributes:

- PayrollID: Integer
- EmployeeID: Integer

- Salary: Double
- Deductions: Double
- Benefits: Double

Operations:

- calculateSalary()
- processDeductions()
- manageBenefits()

HR Manager:

Attributes:

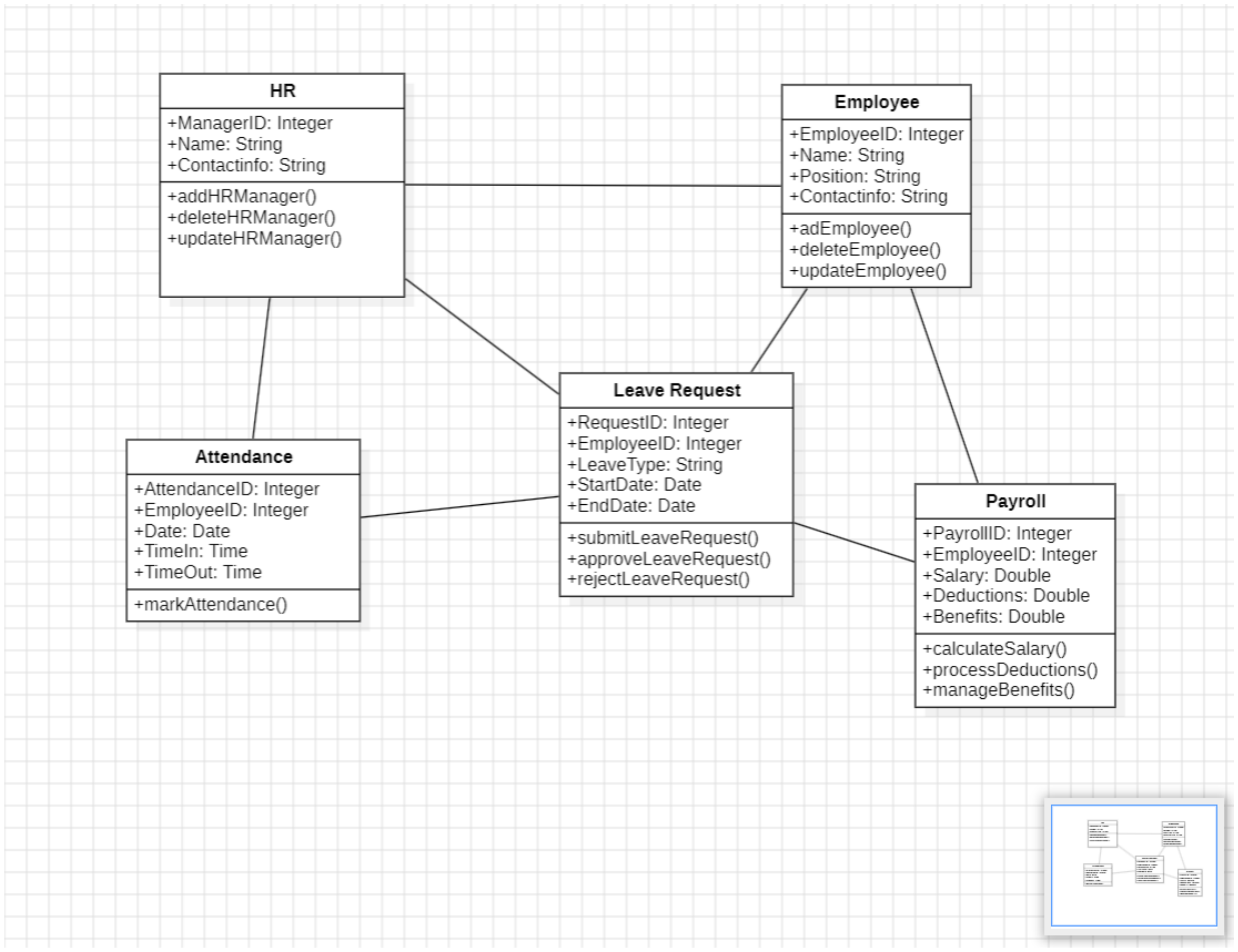
- ManagerID: Integer
- Name: String
- ContactInfo: String

Operations:

- addHRManager()
- deleteHRManager()
- updateHRManager()

The domain model presented above delineates the fundamental entities and their pertinent attributes within the Software Personnel Management System. These entities comprise Employee, Attendance, Leave Request, Payroll, and HR Manager, each equipped with attributes that capture essential information pertinent to the system's operations. Additionally, the model specifies the operations associated with each entity, elucidating the functionalities essential for proficient management of personnel-related tasks and processes. Through these entities and their attributes, the system endeavors to streamline personnel management tasks and enhance operational efficiency within the organization.

Class diagram for Software Personnel management system:



Result:

Thus the class diagram for Software Personnel management system is implemented and executed Successfully.

EX NO: **DATE:**	**Using the Identified Scenarios, Find the Interaction Between Objects and Represent them Using UML Sequence and Collaboration Diagram**

AIM:

To find the interaction between objects and represent them using UML Sequence and Collaboration diagram.

Introduction of Interaction Diagrams:

- **Overview of Interaction Diagrams:**
  1. Interaction diagrams are graphical representations in UML (Unified Modeling Language) used to visualize the dynamic behavior of a system by showing how objects interact over time.
  2. These diagrams capture the flow of messages or actions between objects during the execution of a system or process.
- **Categorization into Two Types:**
  3. Interaction diagrams are categorized into two main types: Activity Diagrams and Sequence Diagrams.
  4. Activity Diagrams: These diagrams focus on modeling the flow of activities or tasks within a system, showing the sequence of actions and decisions.
  5. Sequence Diagrams: These diagrams focus on modeling the chronological order of messages exchanged between objects in a system, illustrating how objects interact over time.

Sequence Diagram:

- A sequence diagram is a type of interaction diagram in UML that represents the interactions between objects over time.
- It illustrates the sequence of messages exchanged between objects within a system to accomplish a specific task or scenario.

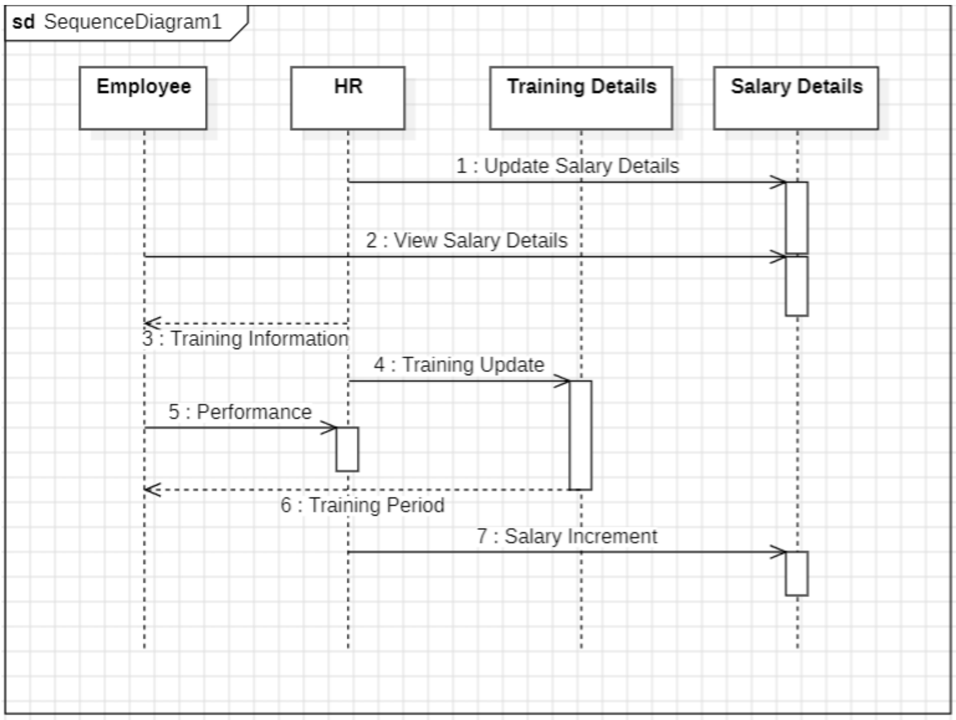
Components of a Sequence Diagram:

1. **Lifelines:**
   - Lifelines represent the lifespan of objects participating in the interaction within the system.
   - Each object involved in the interaction is represented by a lifeline, which is depicted as a vertical dashed line.
   - The length of the lifeline represents the duration of the object's existence or involvement in the interaction.

2. **Messages:**
   - Messages represent communication or interactions between objects in the system.
   - They depict the flow of control or data between objects during the execution of a scenario.
   - Messages are represented by arrows that connect lifelines, indicating the direction of communication.

   - There are different types of messages:
     1. **Synchronous Message:** Represents a direct call or invocation between objects, denoted by a solid arrow.
     2. **Asynchronous Message:** Represents a non-blocking call or invocation between objects, denoted by a dashed arrow.
     3. **Return Message:** Indicates the response or return value from a method call, represented by a dashed arrow with a labeled return value.
     4. **Self-Message:** Represents a message sent from an object to itself, depicted by a looped arrow that returns to the same lifeline.

Sequence Diagram for Software Personnel Management System:



Collaboration Diagram Definition:

A Collaboration Diagram is a type of Interaction Diagram that visualises the interactions and relationships between objects in a system. It shows how objects collaborate to achieve a specific task or behaviour. Collaboration diagrams are used to model the dynamic behaviour of a system and illustrate the flow of messages between objects during a particular scenario or use case.

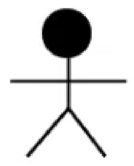
Collaboration Diagram Notation:

1. Objects/Participants:

Object Name

Objects are represented by rectangles with the object's name at the top. The diagram shows each object participating in the interaction as a separate rectangle. Objects are connected by lines to indicate messages being passed between them.

2. Actors:

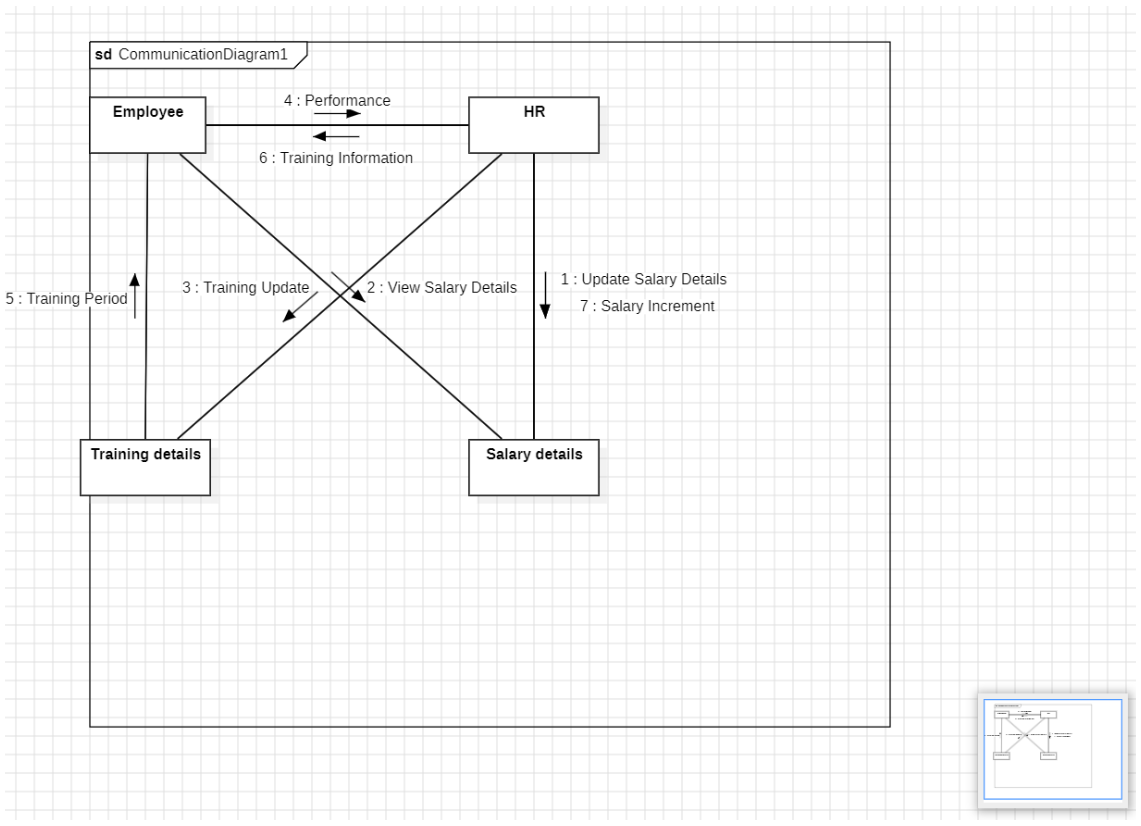


They are usually depicted at the top or side of the diagram, indicating their involvement in the interactions with the system's objects or components. They are connected to objects through messages, showing the communication with the system.

3. Messages:

Messages represent communication between objects. Messages are shown as arrows between objects, indicating the flow of comms. Each message may include a label indicating the type of message. Messages can be asynchronous or synchronous

Communication Diagram For Software Personnel Management System:



Result:

Thus using the identified scenarios, find the interaction between objects and represent them using UML Sequence and Collaboration Diagrams for E-book Management Systems was executed successfully

EX NO: **DATE:**	**Draw Relevant State Chart and Activity Diagram for the Same System**

AIM:

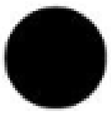
To Draw Activity diagram for Software Personnel Management System

State Chart Diagram Definition:

A State Machine Diagram is used to represent the condition of the system or part of the system at finite instances of time. It's a behavioural diagram and it represents the behaviour using finite state transitions.

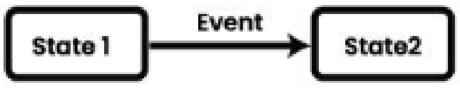
State Chart Diagram Notation:

1. **Initial State:**



We use a black-filled circle to represent the initial state of a System or a Class.

2. **Transition:**

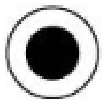


We use a solid arrow to represent the transition or change of control from one state to another. The arrow is labelled with the event which causes the change in state.

3. State:

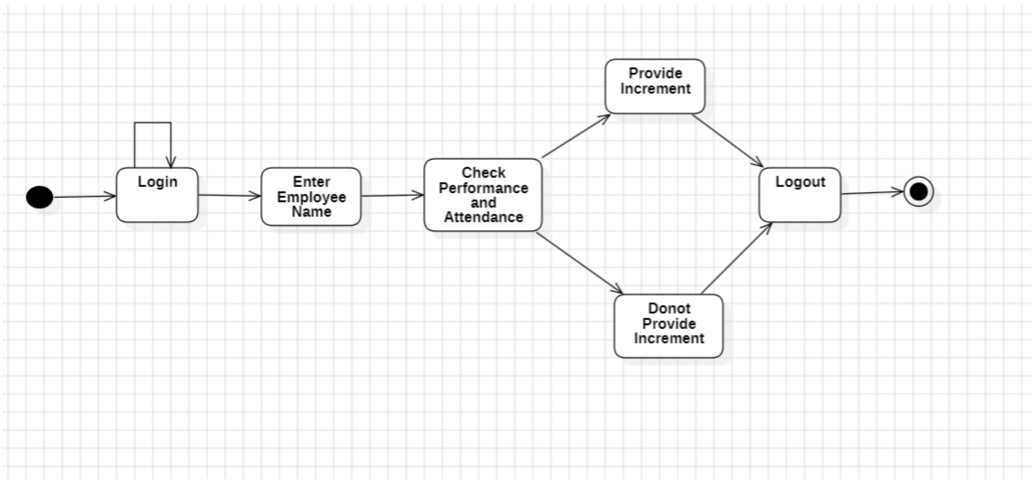
We use a rounded rectangle to represent a state. A state represents the conditions or circumstances of an object of a class at an instant of time

4. **Final State:**



We use a filled circle within a circle notation to represent the final state in a state machine diagram.

State Chart Diagram for Software Personnel Management System:



ACTIVITY DIAGRAM:

1. Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system.
2. An activity diagram shows the overall flow of control. An activity is shown as an rounded box containing the name of the operation. This activity diagram describes the behaviour of the system.
3. An activity diagram is variation or special case or a state machine, in which the states are activities representing a performance of operation and the transition or triggered by the completion of operation.
4. Activity diagram is similar to state chart diagram where the token represented as an operation. An activity shows as a round box containing name of operation. The concurrent control is indicated by multiple arrows, leaving a synchronization bar represented by short or thick bar with incoming and outgoing arrows. Activity diagram is another important diagram in UML to describe dynamic aspects of the system.

Activity Diagram Components:

Initial Node:

Represents the starting point of the activity diagram. It is denoted by a solid circle with an arrow pointing outward.The initial node indicates where the process flow begins within the diagram. It is the first step executed in the sequence of activities.

Final Node:

Represents the ending point of the activity diagram. It is denoted by a solid circle with a border.The final node indicates the completion of the process flow within the diagram. It is the last step executed in the sequence of activities.

Action Node (Activity):

Represents a specific task or operation to be performed within the system. It is denoted by a rectangle with rounded corners.Action nodes represent individual activities or tasks that need to be completed as part of the overall process flow. They encapsulate the actions performed by the system or its components.

Decision Node (Branch):

Represents a branching point based on a condition or decision. It is denoted by a diamond shape.Decision nodes allow the process flow to take different paths based on the evaluation of one or more conditions. They enable conditional behavior within the diagram, where different actions are taken depending on the outcome of a decision.

Merge Node:

Represents the merging of multiple paths of control flow back into a single path. It is denoted by a diamond shape with a single incoming flow and multiple outgoing flows.Merge nodes combine multiple paths of control flow back into a single path. They are used to synchronize parallel branches of the process flow, ensuring that all parallel activities are completed before continuing with the next step.

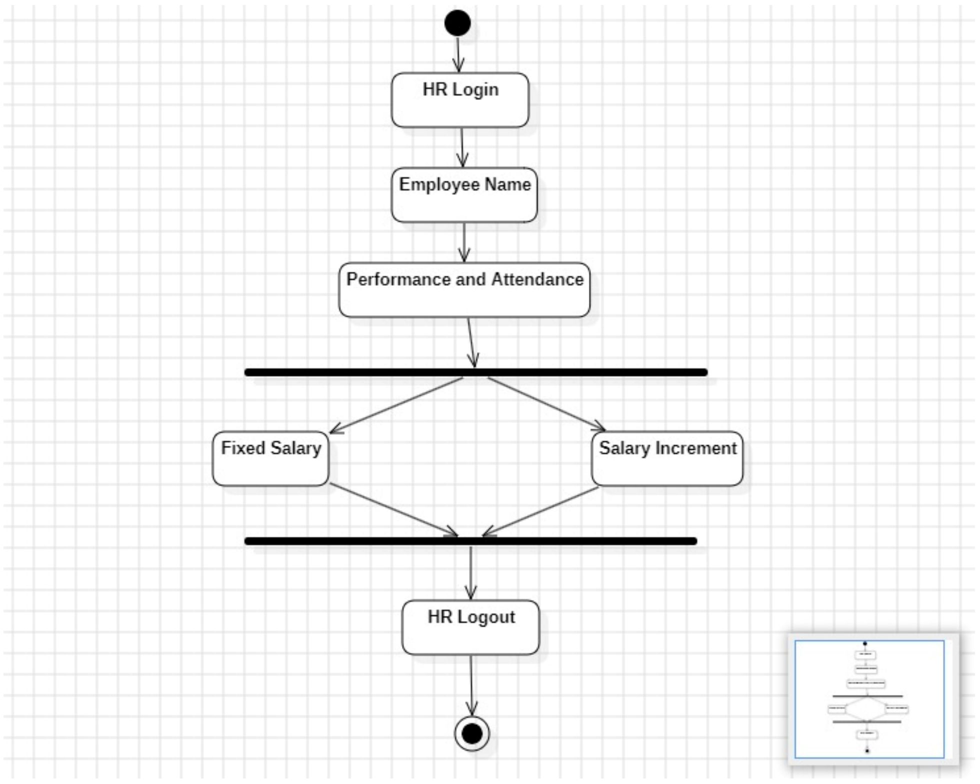
Join Node:

Represents the merging of parallel paths of control flow back into a single path. It is denoted by a bar with multiple incoming flows and a single outgoing flow.Join nodes synchronize parallel branches of the process flow, ensuring that all parallel activities are completed before continuing with the next step. They are used in conjunction with fork nodes to manage parallelism within the diagram.

Control Flow Arrows:

- Control flow arrows indicate the direction of flow between nodes in the activity diagram.
- Solid arrows represent the primary flow of control, while dashed arrows represent alternative or conditional flows.

Activity Diagram For Personnel Management System:



Result:

Thus the State Chart and Activity diagram for Software Personnel Management System was designed successfully.

EX NO: **DATE:**	**Implement the System As Per the Detailed Design**

Aim:

To implement the system as per the detailed design:

Component Diagram Definition:

Component diagrams are used to visualize the organization of system components and the dependency relationships between them. They provide a high-level view of the components within a system.

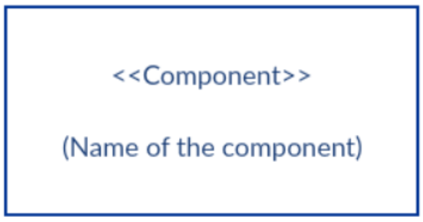
The components can be a software component such as a database or user interface; or a hardware component such as a circuit, microchip or device; or a business unit such as supplier, payroll or shipping.

Component Diagram Notation:

1. Component:

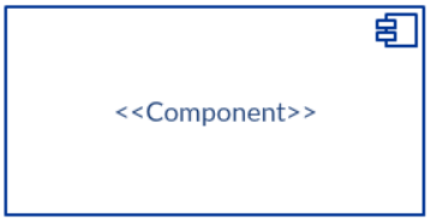
There are three ways the component symbol can be used

a.



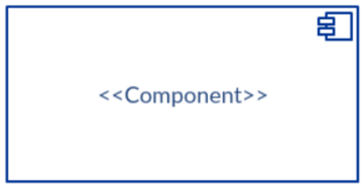
Rectangle with the component stereotype (the text <<component>>). The component stereotype is usually used above the component name to avoid confusing the shape with a class icon.

b.



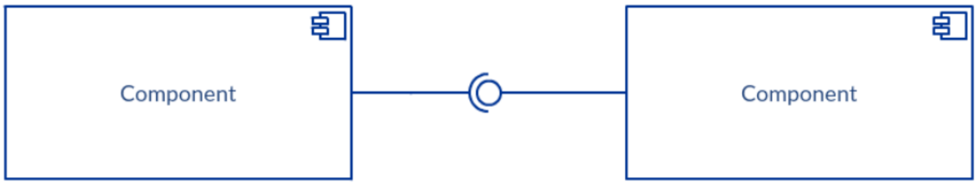
Rectangle with the component icon in the top right corner and the name of the component.

c.



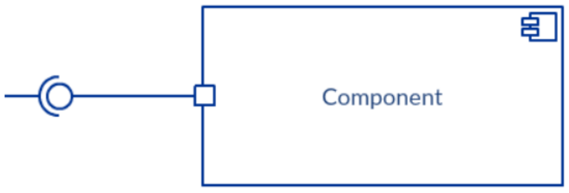
Rectangle with the component icon and the component stereotype.

2. Provided Interface and Required Interface:



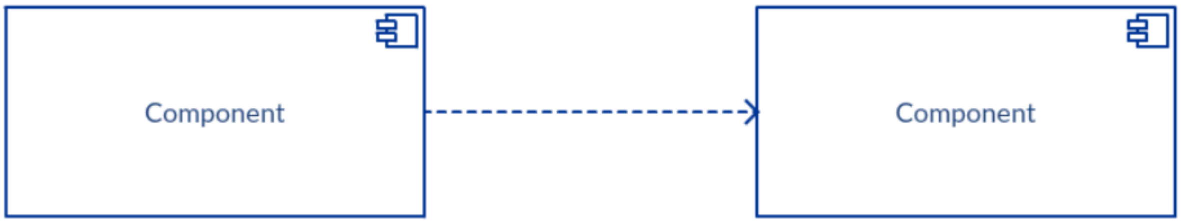
Interfaces in component diagrams show how components are wired together and interact with each other. The assembly connector allows linking the component's required interface (represented by a semi-circle and a solid line) with the provided interface (represented by a circle and solid line) of another component. This shows that one component is providing the service that the other is requiring.

3. Port:



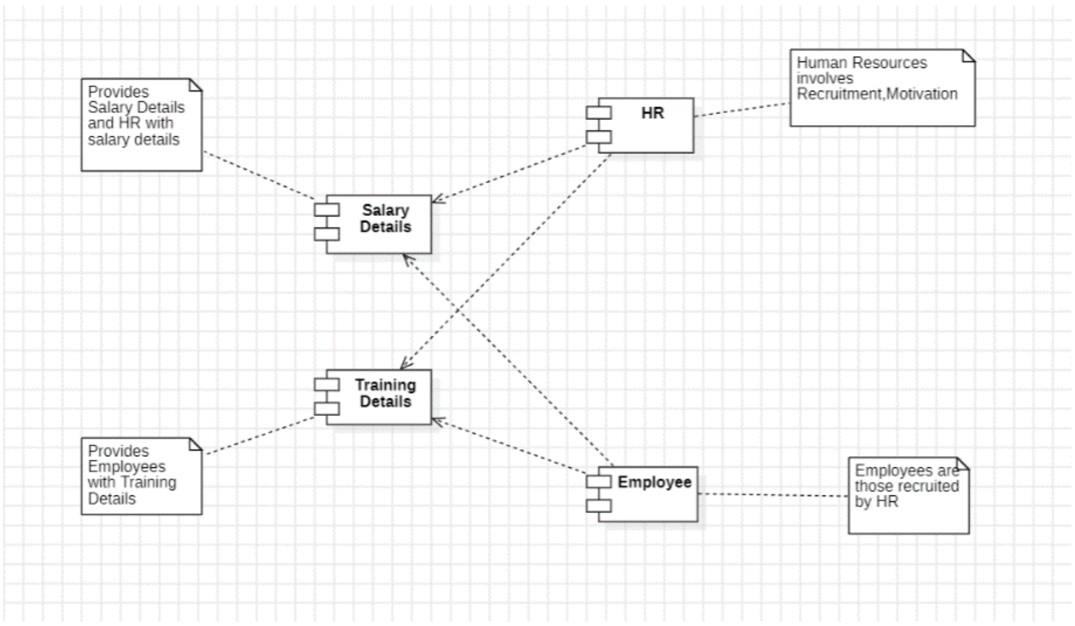
Port (represented by the small square at the end of a required interface or provided interface) is used when the component delegates the interfaces to an internal class.

4. Dependencies:



Although you can show more detail about the relationship between two components using the ball-and-socket notation (provided interface and required interface), you can just as well use a dependency arrow to show the relationship between two components.

Component Diagram:

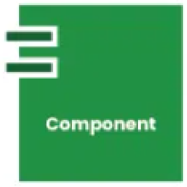

Provides Salary Details and HR with salary details
Human Resources involves Recruitment,Motivation
HR
Salary Details
Training Details
Employee
Provides Employees with Training Details
Employees are those recruited by HR

Deployment Diagram Definition:

A Deployment Diagram in software engineering is a type of Structural UML Diagram that shows the physical deployment of software components on hardware nodes. It illustrates the mapping of software components onto the physical resources of a system, such as servers, processors, storage devices, and network infrastructure.

Deployment Diagram Notation:

1. Component:



A component represents a modular and reusable part of a system, typically implemented as a software module, class, or package. It encapsulates its behaviour and data and can be deployed independently.

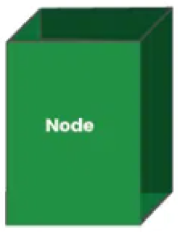
2. **Artifact:**

An artifact represents a physical piece of information or data that is used or produced in the software development process. It can include source code files, executables, documents, libraries, configuration files, or any other tangible item.

3. **Interface:**

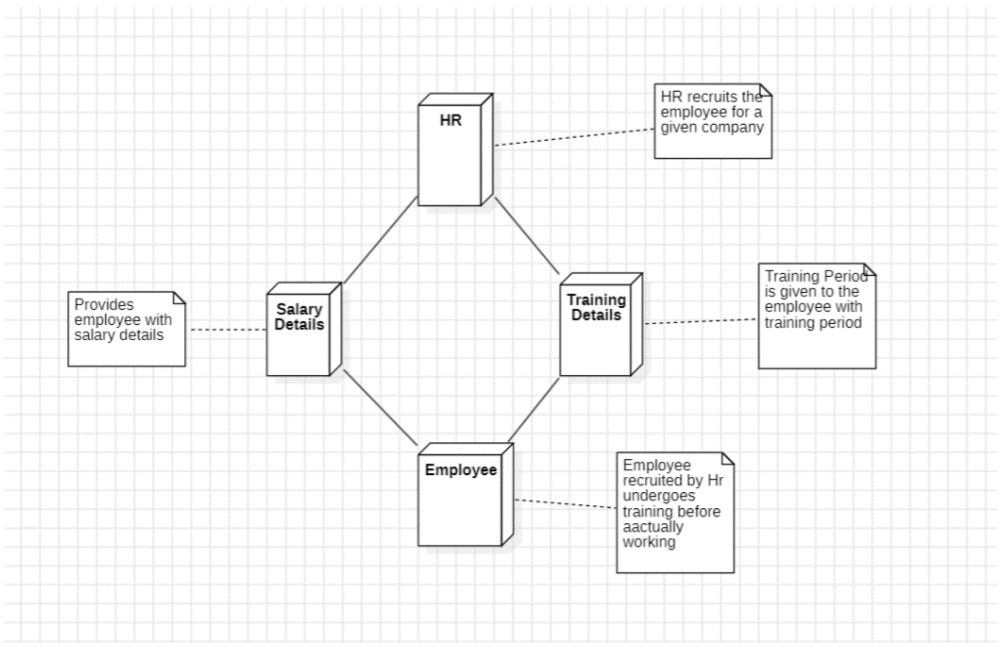
An interface defines a contract specifying the methods or operations that a component must implement. It represents a point of interaction between different components or subsystems.

4. **Node:**



A node represents a physical or computational resource, such as a hardware device, server, workstation, or computing resource, on which software components can be deployed or executed.

Deployment Diagram:


HR
HR recruits the employee for a given company
Provides employee with salary details
Salary Details
Training Details
Training Period is given to the employee with training period
Employee
Employee recruited by Hr undergoes training before aactually working

Implementation For Software Personnel Management System:

Employee.java

```
public class Employee {
   public Employee() {}
   public void Commitment;
   public void Bank Accounts;
   public void Regularity;
   public void Training() {}
   public void Salary() {}
   public void Performance() {}
}
```

HR.java

```
public class HR {
   public HR() {}
   public void Curriculam Vitae;
   public void Monitor Employees;
   public void Provide salary increment;
   public void Motivate;
   public void Recruit() {}
   public void Monitor() {}
   public void Increment() {}
   public void Motivation() {}
}
```

Salary Details.java

```
public class Salary Details extends HR {
   public Salary Details() {}
   public void create bank Account to deposit salary;
   public void salary() {}
}
```

Time mgment.java

```
public class time mgment extends Employee {
   public time mgment() {}
   public void time period observed according to training details;
```

```
    public void time management() {}
}
```

Training Details.java

```
public class Training Details extends HR {
    public Training Details() {}
    public void Provide educated trainees;
    public void Training() {}
}
```

Result:

Thus the implementation of the system as per the detailed design was executed successfully.

EX NO: **DATE:**	**Test the Software System for All Scenarios Identified as Per the Use Case Diagram**

Aim:

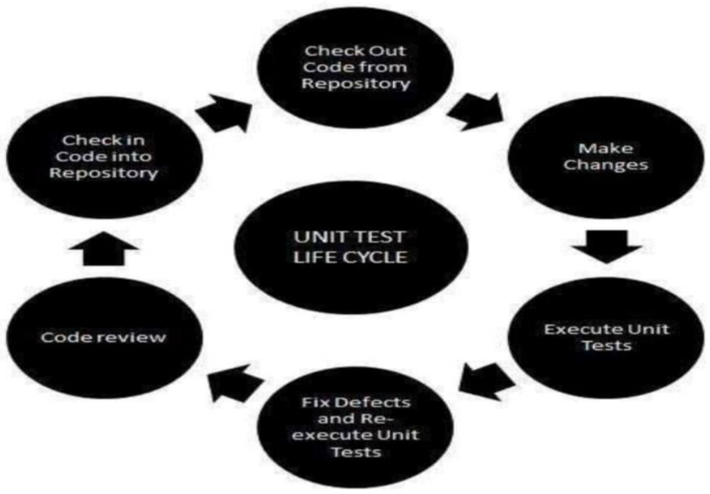
To test the software system for all scenarios identified in the use case diagram.

Software Personnel Management System:

In the domain of software personnel management, efficient handling of human resources is vital for organizational productivity and employee satisfaction. A Software Personnel Management System facilitates the process of managing employee data, assigning tasks, evaluating performance, and streamlining communication within the organization.

i) Unit Testing:

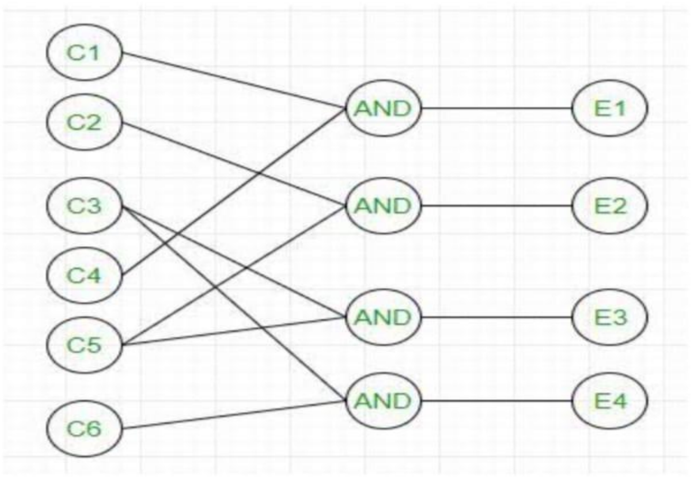
- Unit testing involves testing individual components or modules of the software to ensure they function correctly in isolation.
- Each module, such as employee database management, task allocation, performance evaluation, etc., would undergo unit testing.
- Example: Testing the "addEmployee()" function to ensure that a new employee is successfully added to the system.

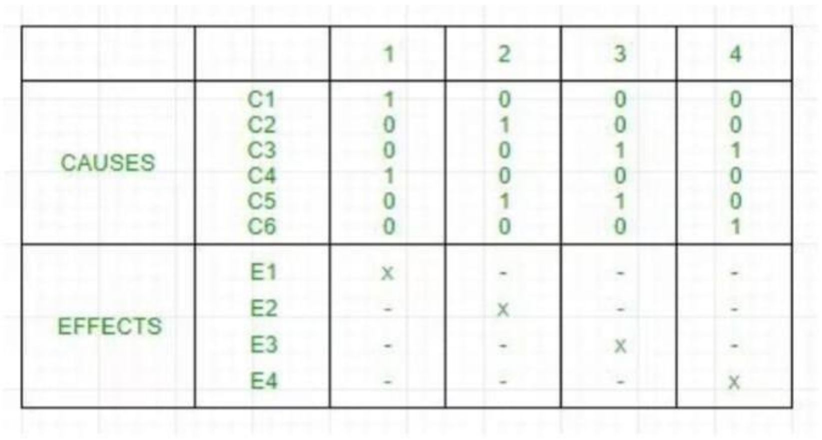

Check Out Code from Repository
Make Changes
Execute Unit Tests
Fix Defects and Re-execute Unit Tests
Code review
Check in Code into Repository
UNIT TEST LIFE CYCLE

ii) Black Box Testing:

- Black box testing is a technique where the internal workings of the system are not known to the tester. The tester only tests the system's functionality based on its specifications.

- Testers input various sets of data into the system and verify that the expected output is produced.
- Example: Testing the "searchForTask()" functionality to ensure that it returns the expected results based on different search criteria.

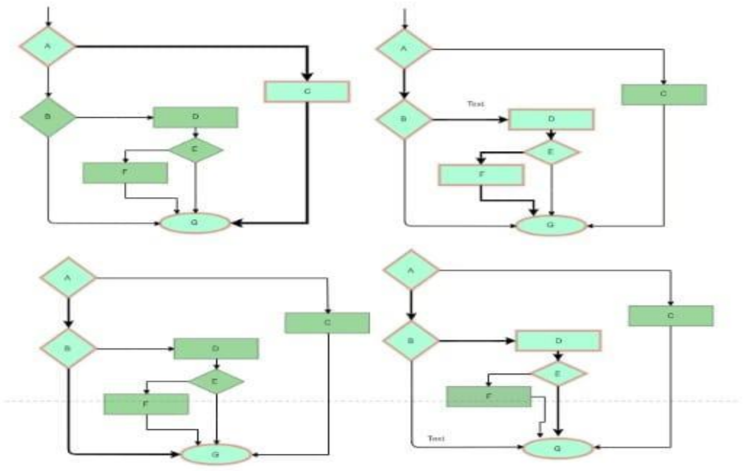




		1	2	3	4
CAUSES	C1	1	0	0	0
	C2	0	1	0	0
	C3	0	0	1	1
	C4	1	0	0	0
	C5	0	1	1	0
	C6	0	0	0	1
EFFECTS	E1	x	-	-	-
	E2	-	x	-	-
	E3	-	-	x	-
	E4	-	-	-	x

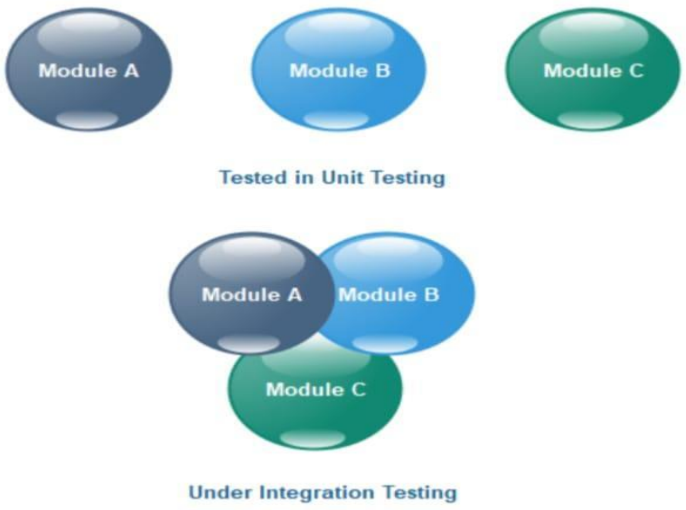
iii) White Box Testing:

- White box testing involves testing the internal logic and structure of the software code.
- Testers examine the code of individual modules to ensure that all code paths are tested.
- Example: Testing the "evaluatePerformance()" function to ensure that it adequately calculates employee performance metrics.



iv) Integration Testing:

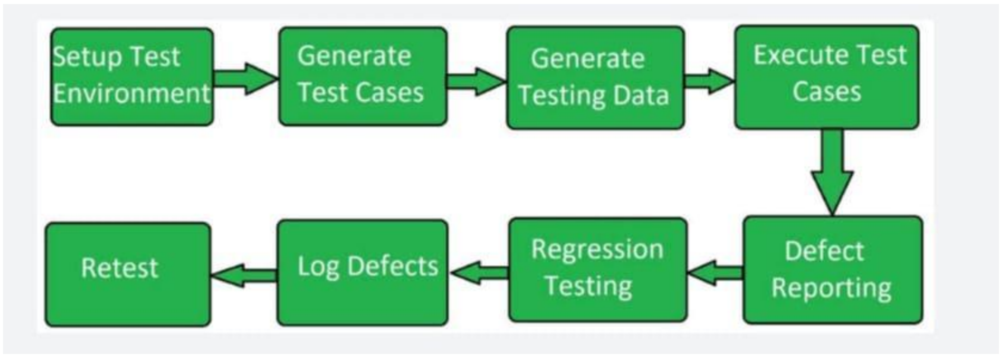
- Integration testing verifies that different modules of the software work together as expected.
- Testers test the interaction between different modules, such as employee database management, task allocation, and performance evaluation.
- Example: Testing the interaction between the "addEmployee()" and "assignTask()" functions to ensure that tasks can be assigned to employees successfully.



v) System Testing:

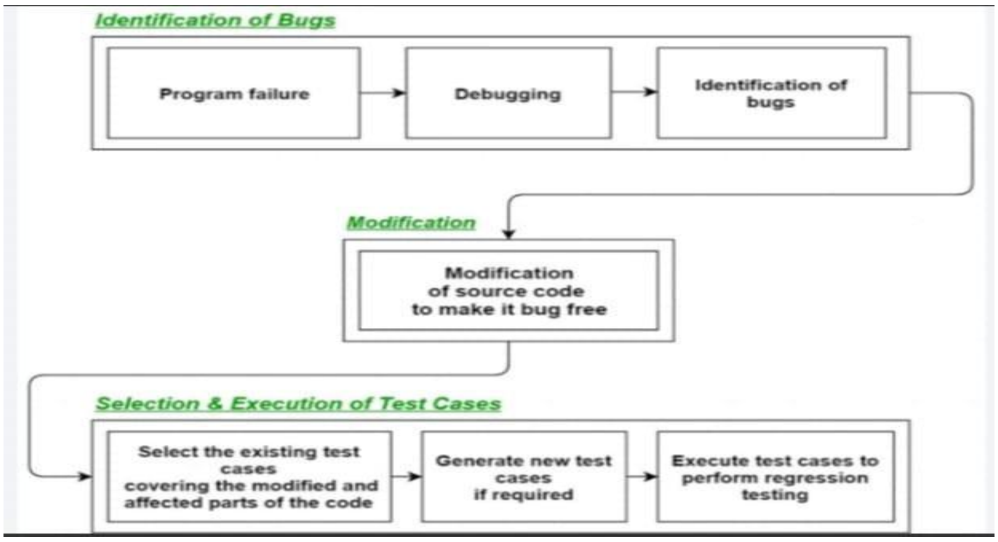
- System testing involves testing the entire system as a whole to verify that it meets the specified requirements.

- Testers perform end-to-end testing of the entire system, including all modules and their interactions.
- Example: Testing the entire process from employee onboarding to task completion and performance review to ensure that it functions as expected.



vi) Regression Testing:

- Regression testing ensures that new changes or additions to the system do not adversely affect existing functionalities.
- Testers re-run previously conducted tests after new changes or additions are made to the system to ensure that existing functionalities are not affected.
- Example: After adding a new feature to the system, testers would re-run all existing tests to ensure that the new feature did not introduce any bugs or errors.


Identification of Bugs
Program failure
Debugging
Identification of bugs
Modification
Modification of source code to make it bug free
Selection & Execution of Test Cases
Select the existing test cases covering the modified and affected parts of the code
Generate new test cases if required
Execute test cases to perform regression testing

Result:

Thus the software system for all scenarios identified in the use case diagram was tested successfully.

EX NO: DATE:	**Improve the reusability and maintainability of the software by applying appropriate design patterns**

Aim:

To improve the reusability and maintainability of the software system by applying appropriate design patterns.

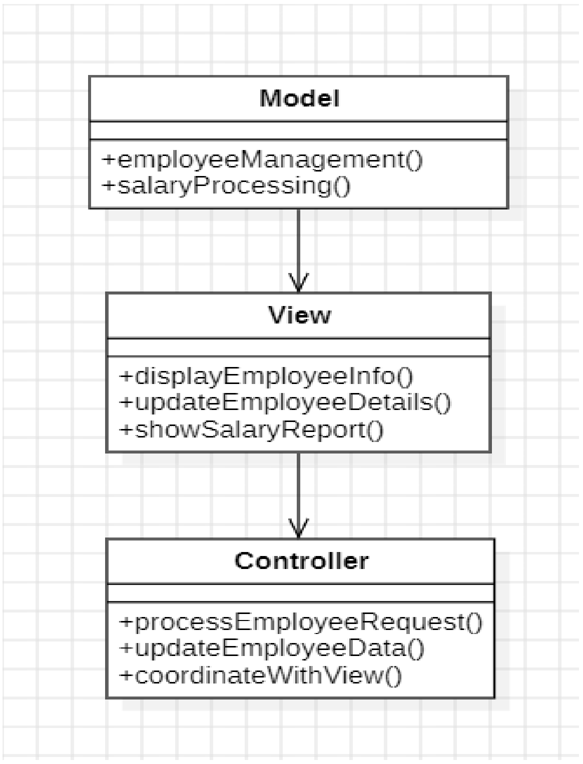
Software Personnel Management System:

A Software Personnel Management System facilitates the efficient management of human resources within an organization. It includes functionalities such as employee data management, task allocation, performance evaluation, and communication.

To enhance the reusability and maintainability of the BPO management system, we can incorporate several design patterns. Here are some design patterns that can be applied:

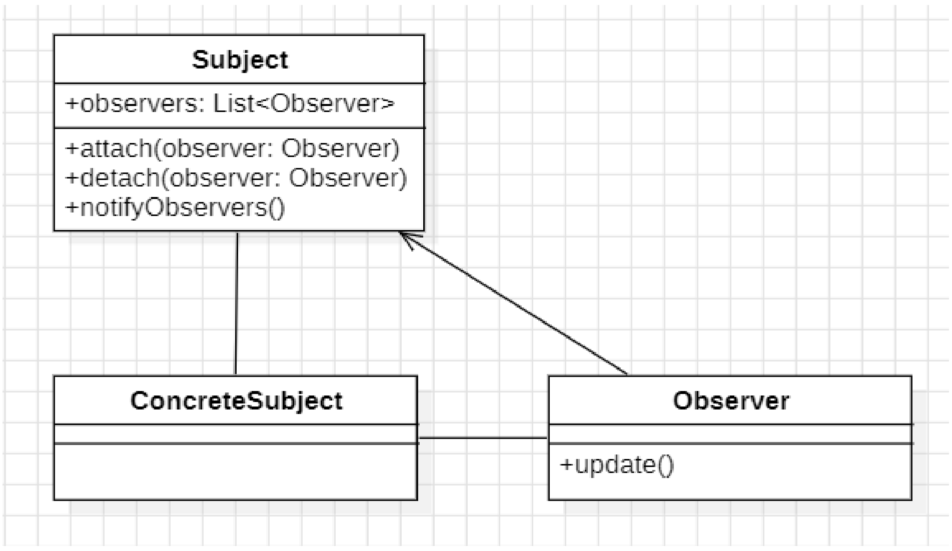
i) Model-View-Controller (MVC) Pattern:

The MVC pattern separates the representation of information from the user's interaction with it. In the Software Personnel Management System, it divides the system into three interconnected components: Model (data and business logic), View (user interface), and Controller (handles user input and updates the model and view accordingly).



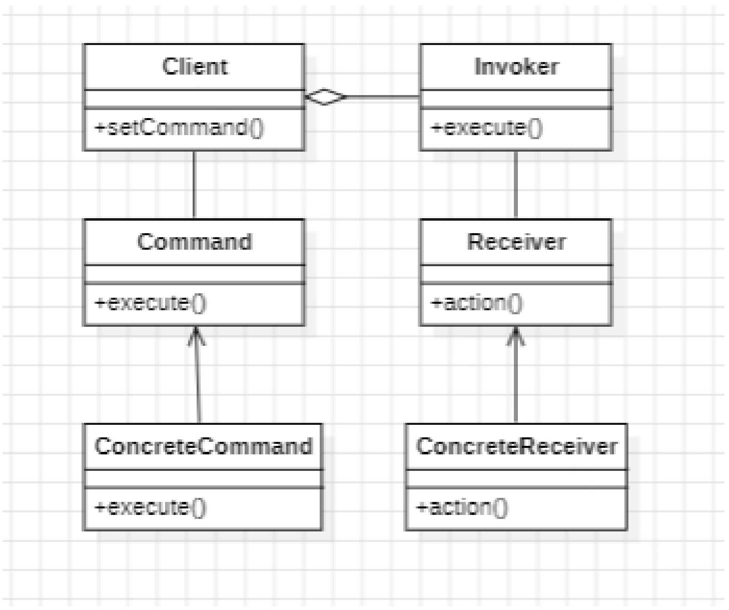
ii) Observer Pattern:

The Observer pattern establishes a one-to-many dependency between objects, where multiple observers are notified of changes in a subject. In the Software Personnel Management System, it can be used to notify stakeholders about important events such as employee onboarding, task assignments, or performance reviews.


Subject
+observers: List<Observer>
+attach(observer: Observer)
+detach(observer: Observer)
+notifyObservers()
ConcreteSubject
Observer
+update()

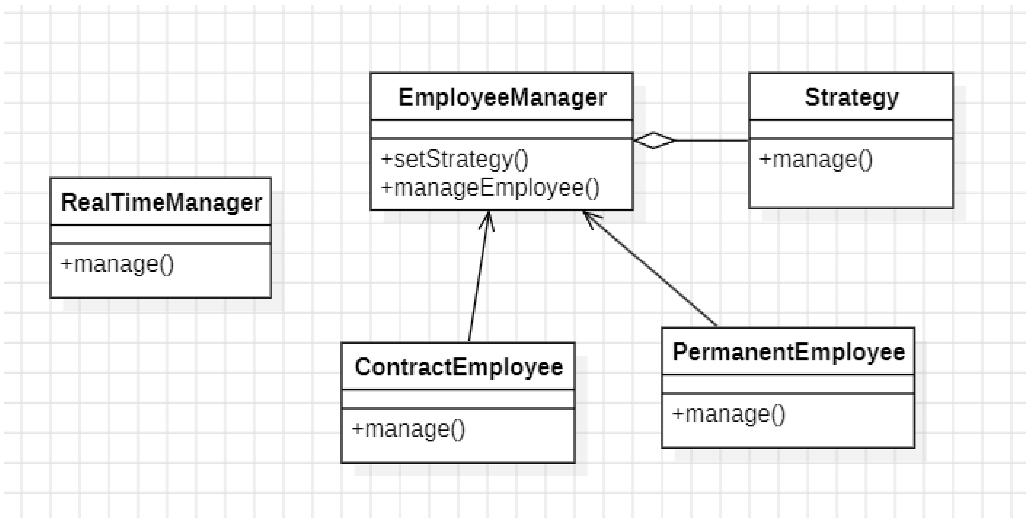
iii) Command Pattern:

The Command pattern encapsulates requests as objects, allowing for parameterization of clients with queued or logged requests. In the Software Personnel Management System, it can be used to implement features like undo/redo functionality for task assignments, batch processing of employee data updates, or logging of administrative actions.



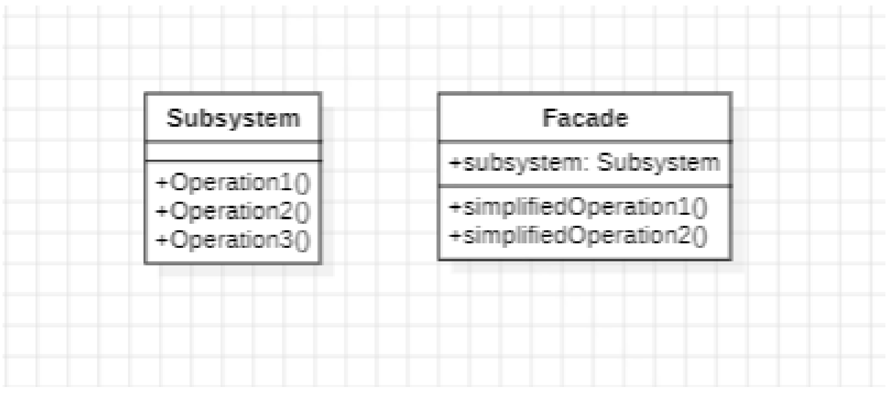
iv) Strategy Pattern:

The Strategy pattern defines a family of algorithms, encapsulates each one, and makes them interchangeable. In the Software Personnel Management System, it can be used to implement different strategies for task prioritization, performance evaluation, or communication protocols based on organizational needs.



v) Facade Pattern:

The Facade pattern provides a simplified interface to a complex subsystem, hiding its complexities from clients. In the Software Personnel Management System, it can be used to abstract away the complexities of employee data management, task allocation, or performance evaluation, providing a unified interface for interacting with these subsystems.



Result:

Thus the reusability and maintainability of the software system by applying appropriate design patterns was executed successfully.