**EXPERIMENT : 1**

**Title :** Basics of OOAD.

Object-Oriented Analysis and Design (OOAD) is a method for analyzing and designing a system by visualizing it as a group of interacting objects, each defined by its attributes (data) and methods (functions). The core elements of OOAD include several key concepts, such as classes, objects, inheritance, polymorphism, encapsulation, and abstraction. Here’s a description of each of these elements with examples:

**1. Class**

A class is a blueprint or template for creating objects. It defines the properties and behaviors (attributes and methods) common to all objects of that type.

**Example:** Imagine you are building a system for a library. You might have a Book class that defines properties like title, author, isbn, and methods like checkout(), returnBook().

class Book {

String title;

String author;

String isbn;

void checkout() {

// Code for checking out the book

}

void returnBook() {

// Code for returning the book

}

}

**2. Object**

An object is an instance of a class. While a class is a blueprint, an object is a specific instance that follows that blueprint.

**Example:** From the Book class above, we can create an object representing a specific book.

Book book1 = new Book();

book1.title = "1984";

book1.author = "George Orwell";

book1.isbn = "1234567890";

book1.checkout(); // Calling method

**3. Encapsulation**

Encapsulation is the concept of restricting access to certain details of an object’s data and hiding the complexity from the outside world. This is typically achieved by making attributes private and providing public getter and setter methods to access or modify them.

**Example:** In the Book class, you might want to hide the details of the isbn and provide a getter and setter.

class Book {

private String isbn;

public String getIsbn() {

return isbn;

}

public void setIsbn(String isbn) {

this.isbn = isbn;

}

}

Here, you’re encapsulating the isbn field and providing controlled access to it.

**4. Abstraction**

Abstraction refers to hiding complex implementation details and showing only the essential features of an object. In OOAD, you define abstract classes or interfaces that provide a common structure without specifying the detailed implementation.

**Example:** Consider an abstract class Shape that defines the method draw() but doesn’t specify how it is drawn.

abstract class Shape {

abstract void draw(); // Abstract method

}

class Circle extends Shape {

@Override

void draw() {

// Circle-specific drawing code

}

}

class Square extends Shape {

@Override

void draw() {

// Square-specific drawing code

}

}

**5. Inheritance**

Inheritance is a mechanism where a new class (subclass or derived class) inherits the attributes and methods of an existing class (superclass or base class). It allows for code reuse and establishing a relationship between different classes.

**Example:** In the Shape example, Circle and Square both inherit from the Shape class. This allows them to share a common draw() method while implementing their own specific behaviors.

class Circle extends Shape {

// Inherits draw() from Shape

}

class Square extends Shape {

// Inherits draw() from Shape

}

**6. Polymorphism**

Polymorphism means "many forms." It allows objects of different classes to be treated as objects of a common superclass. The two main types of polymorphism are:

* **Method Overloading:** Defining multiple methods with the same name but different parameters.
* **Method Overriding:** Redefining a method in a subclass to provide specific behavior.

**Example:** In the Shape example, polymorphism allows you to call draw() on any Shape object, and it will call the correct version based on the actual object type.

Shape shape1 = new Circle();

Shape shape2 = new Square();

shape1.draw(); // Calls Circle's draw()

shape2.draw(); // Calls Square's draw()

**Conclusion**

OOAD is a powerful methodology for designing systems by focusing on the objects and their interactions. These key elements—classes, objects, inheritance, encapsulation, abstraction, polymorphism help developers model real-world entities and their behaviors in a way that is flexible, reusable, and maintainable.

**EXPERIMENT : 2**

**Title :** Write a Problem Statement.

**Problem Statement for the Mini Project: AgroSense**

**Project Title:** AgroSense - Agricultural Data Analysis and Prediction for Madhya Pradesh (MP), India

**Objective:** The objective of the AgroSense mini project is to develop a data-driven system that assists farmers, agricultural enthusiasts, and policymakers by providing valuable insights into various aspects of agriculture in Madhya Pradesh (MP), India. The system will focus on analyzing the relationship between soil types, crop types, weather conditions, and their impact on agricultural productivity. Additionally, the project will include real-time weather data and price trends of different crops, helping farmers make informed decisions about their crops.

**Features of the AgroSense Project:**

1. **Soil Data Analysis**:
   * This feature will provide information about the different types of soils present in Madhya Pradesh, India, and their percentage distribution across the region. The system will classify soil types based on properties such as texture, pH, and nutrients, and display how much area of MP is covered by each type.
2. **Crop Data Analysis**:
   * The system will display data on the different types of crops grown in Madhya Pradesh and their percentage of cultivation across the state. It will include major crops like wheat, rice, soybeans, and pulses, as well as emerging crops. This feature will allow users to gain insights into the cropping pattern and distribution.
3. **Soil vs Crop Data**:
   * This feature will present a relationship between different soil types and the crops that thrive in those conditions. Based on the soil data, the system will suggest which crops perform best in each soil type, providing valuable information for optimizing crop yield in various regions of MP.
4. **Crop vs Weather Data**:
   * This feature will help farmers identify the ideal weather conditions (temperature, rainfall, humidity, etc.) for growing specific crops. By correlating historical weather data with crop yields, this section will recommend the best seasons and climatic conditions for different crops in Madhya Pradesh.
5. **Current Weather Condition for MP**:
   * Using real-time weather data, this feature will provide current weather conditions in different parts of Madhya Pradesh. It will include details like temperature, humidity, rainfall, wind speed, and other relevant parameters, allowing farmers to adjust their agricultural activities accordingly.
6. **Crop Price Data**:
   * This feature will present the current market prices of various crops grown in Madhya Pradesh. It will provide up-to-date price trends for different crops in the region, enabling farmers to make better financial decisions regarding which crops to grow based on potential profitability.

**Use Case Scenario:**

* **Farmers** can use the system to identify which crops will perform best in their region, based on the soil type and weather conditions. They can also monitor current weather conditions and crop prices to maximize profitability.
* **Agricultural researchers** can use the platform to analyze the relationship between soil, crops, and weather, helping to identify optimal farming practices for different regions of MP.
* **Government and policymakers** can use the insights from the system to formulate strategies for enhancing agricultural productivity, ensuring food security, and improving the livelihoods of farmers.

**Technologies Used:**

* Data collection and processing tools (e.g., APIs for weather data, agriculture databases).
* Data visualization techniques to present results in the form of graphs, charts, and maps.
* Real-time weather tracking and prediction algorithms.
* A user-friendly interface for easy navigation and interaction with the data.

By integrating these features, AgroSense aims to be a comprehensive platform for improving agricultural productivity and supporting sustainable farming practices in Madhya Pradesh.

**EXPERIMENT : 3**

**Title :** Perform the system analysis: Requirement analysis, SRS.

**Software Requirements Specification**

**for**

**<AGROSENSE >**

**Version 1.0 approved**

**Prepared by <author>**

**<organization>**

**<date created>Table of Contents**

**Table of Contents** ii  
**Revision History** ii

**1. Introduction**

1.1 Purpose  
1.2 Document Conventions  
1.3 Intended Audience and Reading Suggestions  
1.4 Product Scope  
1.5 References

**2. Overall Description**

2.1 Product Perspective  
2.2 Product Functions  
2.3 User Classes and Characteristics  
2.4 Operating Environment  
2.5 Design and Implementation Constraints  
2.6 User Documentation  
2.7 Assumptions and Dependencies

**3. External Interface Requirements**

3.1 User Interfaces  
3.2 Hardware Interfaces  
3.3 Software Interfaces  
3.4 Communications Interfaces

**4. System Features**

4.1 Crop Data Module  
4.2 Price Distribution Module  
4.3 Weather Forecasting Module  
4.4 Soil & Crop Suitability Module

**5. Other Nonfunctional Requirements**

5.1 Performance Requirements  
5.2 Safety Requirements  
5.3 Security Requirements  
5.4 Software Quality Attributes  
5.5 Business Rules

**6. Other Requirements**

**Appendices**

Appendix A: Glossary  
Appendix B: Analysis Models  
Appendix C: To Be Determined List

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
|  |  |  |  |
|  |  |  |  |

**1. Introduction**

**1.1 Purpose**

This document defines the requirements for the **Agriculture Decision Support System (ADSS)**, designed to assist farmers, policymakers, and agribusinesses in Madhya Pradesh (MP), India. The system provides data on crops, market prices, weather forecasts, and soil suitability to improve agricultural decision-making.

**1.2 Document Conventions**

* **Bold**: Key terms and system features.
* *Italics*: References to external documents.
* Code: API endpoints or technical terms.

**1.3 Intended Audience and Reading Suggestions**

* **Farmers**: Focus on Sections 4 (System Features) and 3.1 (User Interfaces).
* **Developers**: Review Sections 3 (Interfaces) and 5 (Nonfunctional Requirements).
* **Government Agencies**: Refer to Sections 2.2 (Product Functions) and 4.2 (Price Distribution).

**1.4 Product Scope**

ADSS is a **web/mobile-based platform** that:

* Provides **crop production data** for MP.
* Tracks **real-time and historical crop prices**.
* Integrates **weather forecasting** for farm planning.
* Recommends **crops based on soil and weather**.

**1.5 References**

1. *MP Agriculture Department Reports*
2. *IMD (India Meteorological Department) API Documentation*
3. *NBSS&LUP Soil Data Guidelines*

**2. Overall Description**

**2.1 Product Perspective**

ADSS is a standalone system but integrates with:

* Weather APIs (e.g., IMD, OpenWeatherMap).
* Government agriculture databases (e.g., Agmarknet for prices).

**2.2 Product Functions**

1. **Crop Data Module**: Displays crop yields, seasons, and district-wise trends.
2. **Price Distribution Module**: Shows market prices and trends.
3. **Weather Module**: Forecasts and alerts for extreme weather.
4. **Soil & Suitability Module**: Recommends crops based on soil type.

**2.3 User Classes and Characteristics**

| **User Class** | **Characteristics** |
| --- | --- |
| Farmers | Low technical expertise; needs simple UI. |
| Agri-Traders | Requires real-time price data. |
| Government | Needs exportable reports for policy analysis. |

**2.4 Operating Environment**

* **Frontend**:
* **Backend**:
* **Database**:
* **APIs**: Weather (IMD), Soil (NBSS&LUP), Prices (Agmarknet)

**2.5 Design and Implementation Constraints**

* Support  **English**.

Only online functionality for the areas.

**2.6 User Documentation**

* **User Manual**: Step-by-step guides for farmers.
* **API Documentation**: For developers integrating weather/price data.

**2.7 Assumptions and Dependencies**

* Assumes **stable internet** in rural MP (fallback: SMS alerts).
* Depends on **third-party APIs** (IMD, Agmarknet).

**3. External Interface Requirements**

**3.1 User Interfaces**

* **Dashboard**: Maps, charts for crop/price/weather data.
* **Mobile App**: Simplified UI for farmers.

**3.2 Hardware Interfaces**

* Compatible with **low-end Android devices**.

**3.3 Software Interfaces**

* **Weather API**: https://api.imd.gov.in
* **Soil API**: NBSS&LUP’s public dataset.

**3.4 Communications Interfaces**

* SMS gateways for weather alerts.

**4. System Features**

**4.1 Crop Data Module**

* Displays **crop production** (wheat, soybean, pulses) by district.
* **Filterable** by season (Kharif/Rabi).

**4.2 Price Distribution Module**

* **Real-time prices** from APMC markets.
* **Historical trends** (5+ years).

**4.3 Weather Forecasting Module**

* **Forecasts** (rainfall, temperature).
* **Alerts** for droughts/floods.

**4.4 Soil & Crop Suitability Module**

* **Soil maps** (black, alluvial, red).
* **Crop recommendations** (e.g., black soil → soybean).

**5. Other Nonfunctional Requirements**

**5.1 Performance Requirements**

* Load data within **few seconds**.

**5.2 Safety Requirements**

* **Data validation** to prevent incorrect crop recommendations.

**5.3 Security Requirements**

* **User authentication**.

**5.4 Software Quality Attributes**

* **Reliability**: 99% uptime.

**5.5 Business Rules**

* Price data updates **daily** from Agmarknet.

**6. Other Requirements**

**.** Some more data about the various states for future enhancement

**Appendices**

**Appendix A: Glossary**

* **APMC**: Agricultural Produce Market Committee.
* **NBSS&LUP**: National Bureau of Soil Survey & Land Use Planning.

**Appendix B: Analysis Models**

* **Data Flow Diagrams (DFDs)** for weather/crop modules.

**Appendix C: To Be Determined List**

* **TBD**: Drone-based field monitoring feasibility.

**EXPERIMENT : 4**

**Title :** To study about the Use Case Model and Gantt chart.

Absolutely! Here's a more detailed explanation of the Use Case Model, while still keeping it clear and not too overwhelming:

**Use Case Model**

**Definition:**

A Use Case Model is a tool used in software engineering and system design to capture the functional requirements of a system. It describes how users interact with a system to accomplish specific goals, focusing on what the system does rather than how it does it.

**Core Elements:**

**1. Actors:**

* These are users or external systems that interact with the system.
* They can be:
  + Primary Actors – initiate the interaction (e.g., Customer, Faculty).
  + Secondary Actors – provide a service (e.g., Bank System, Database).

**2. Use Cases:**

* Describe a specific function or goal the actor wants to achieve using the system.
* Each use case represents a scenario of interaction between the actor and the system.
* Examples:
  + "Submit Lab Report"
  + "Generate Marksheet"
  + "Login"

**3. System Boundary:**

* A box drawn around the use cases to show what is part of the system and what is outside.
* Actors are outside the boundary; use cases are inside.

**4. Relationships Between Use Cases:**

* **Association:** Direct interaction between actor and use case.
* **Include:** One use case always calls another.
  + Example: "Submit Report" includes "Login"
* **Extend:** One use case optionally adds to another.
  + Example: "Generate Report" might extend "View Report Details"
* **Generalization:** Inheritance between actors or use cases.
  + Example: "Admin" and "Instructor" inherit from a generic "User".

**Purpose of a Use Case Model:**

1. Capture functional requirements in a user-friendly way.
2. Bridge the gap between stakeholders and technical teams.
3. Helps in:
   * Planning system behavior
   * Designing user interfaces
   * Writing test cases
4. Promotes clarity and scope definition early in development.

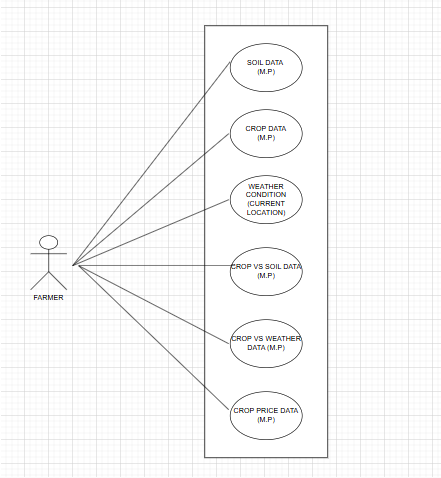
**Visual Representation (UML Use Case Diagram):**

A UML (Unified Modeling Language) Use Case Diagram is used to draw the model. It includes:

* Ovals for use cases
* Stick figures for actors
* Arrows/lines showing interactions

**Example Scenario: Online Course Portal**

| **Actor** | **Use Case** |
| --- | --- |
| Student | Register, View Courses, Submit Assignment |
| Instructor | Upload Materials, Grade Assignment |
| Admin | Add Users, Manage Courses |



## ****Gantt Chart:****

A **Gantt Chart** is a **visual project management tool** used to plan, schedule, and track tasks or activities over time. It shows **what needs to be done, when it needs to be done,** and **how long it will take.**

### ****Basic Components of a Gantt Chart:****

1. **Tasks/Activities:**
   * Listed on the vertical (Y-axis).
   * Each task represents a unit of work, like "Write Lab Manual" or "Get Approval".
2. **Timeline:**
   * Displayed on the horizontal (X-axis).
   * Shows the time span of the project (days, weeks, or months).
3. **Bars:**
   * Horizontal bars represent the **duration of each task**.
   * The **start and end dates** of the bars indicate the task timeline.
4. **Dependencies (optional):**
   * Arrows or lines between tasks show **relationships** (e.g., Task B starts after Task A ends).

### ****Purpose of a Gantt Chart:****

* Helps in **project planning** and setting deadlines.
* Shows **task overlaps** and **sequences**.
* Makes it easy to track **progress** over time.
* Useful for identifying **bottlenecks** or **delays**.

### ****Use of Gantt Chart :****

| **Benefit** | **Description** |
| --- | --- |
| Visual Clarity | Easy to understand task schedules. |
| Time Management | Tracks deadlines and durations. |
| Team Collaboration | Assigns and coordinates team roles. |
| Task Dependency Tracking | Shows how tasks are linked. |
| Progress Monitoring | Updates on how much work is completed. |

### ****Example: Lab Manual Preparation Project (Gantt View)****

| **Task** | **Start Date** | **End Date** | **Duration** |
| --- | --- | --- | --- |
| Choose Topics | Apr 1 | Apr 3 | 3 days |
| Write Theory Content | Apr 4 | Apr 10 | 7 days |
| Create Diagrams | Apr 5 | Apr 8 | 4 days |
| Proofread & Review | Apr 11 | Apr 13 | 3 days |
| Submit for Approval | Apr 14 | Apr 15 | 2 days |
| Final Print or Upload | Apr 16 | Apr 17 | 2 days |

Each of these tasks would be a horizontal bar on the chart spanning the number of days shown above.

