###### Untitled-2

###### University of the Punjab

###### Gujranwala Campus

**First Deliverable**

**BSE-2114**

Table of Contents

[First Deliverable Guide 2](#_Toc189323177)

[1 Introduction 2](#_Toc189323178)

[1.1 Project/Product Feasibility Report 3](#_Toc189323179)

[1.1.1 Technical Feasibility 3](#_Toc189323180)

[1.1.2 Operational Feasibility 4](#_Toc189323181)

[1.1.3 Economic Feasibility 4](#_Toc189323182)

[1.1.4 Schedule Feasibility 4](#_Toc189323183)

[1.1.5 Specification Feasibility 4](#_Toc189323184)

[1.1.6 Information Feasibility 5](#_Toc189323185)

[1.1.7 Motivational Feasibility 5](#_Toc189323186)

[1.1.8 Legal & Ethical Feasibility 5](#_Toc189323187)

[1.2 Project/Product Scope 5](#_Toc189323188)

[1.3 Project/Product Costing 6](#_Toc189323189)

[1.3.1 Project Cost Estimation By Function Point Analysis 7](#_Toc189323190)

[1.3.2 Project Cost Estimation by using COCOMO’81 (Constructive Cost Model) 10](#_Toc189323191)

[1.4 CPM - Critical Path Method 13](#_Toc189323192)

[1.5 Gantt chart 17](#_Toc189323193)

[1.6 Introduction to Team member and their skill set 17](#_Toc189323194)

[1.7 Tools and Technology with Reasoning 18](#_Toc189323195)

[1.8 Vision Document 22](#_Toc189323196)

[1.9 Risk List 24](#_Toc189323197)

[1 Introduction 26](#_Toc189323198)

[1.1 Requirement Elicitation 26](#_Toc189323199)

[1.2 Requirement Analysis and Negotiation 27](#_Toc189323200)

[1.1 Systems Specifications 29](#_Toc189323201)

[1.2 Identifying External Entities 32](#_Toc189323202)

[1.3 Context Level Data Flow Diagram 32](#_Toc189323203)

[1.3 Capture "shall" Statement 33](#_Toc189323204)

[1.5 Allocate Requirements 34](#_Toc189323205)

[1.6 Prioritize Requirements 35](#_Toc189323206)

[1.7 Requirements Trace-ability Matrix 37](#_Toc189323207)

[1.9 High Level Usecase Diagram 39](#_Toc189323208)

# First Deliverable Guide

## 1 Introduction

The first part of this deliverable is all about planning and scheduling of project. This deliverable must contain the following artifacts:

1. Project Feasibility
2. Project Scope
3. Project Costing
4. Critical Path Method Analysis (CPM Analysis)
5. Gantt Chart
6. Introduction to team members
7. Tools and Technologies
8. Vision Document
9. Risk List

## 1.1 Project/Product Feasibility Report

The first step in starting the **Plant Disease Detection System** project is conducting a feasibility study. This process evaluates whether the project is practical and beneficial. It assesses the availability of essential resources, such as the required staff, technical skills, time, and equipment, to successfully build the system. Furthermore, it investigates the availability and accessibility of critical data, such as plant images or disease datasets, and the tools necessary for developing the system, like machine learning models and image processing techniques. The feasibility study assesses if the Plant Disease Detection System can be developed with available resources and effectively solve plant disease detection. It ensures the project stays on track and can be adjusted as needed throughout its lifecycle.

There are many types of feasibilities:

* Technical
* Operational
* Economic
* Schedule
* Specification
* Information
* Motivational
* Legal and Ethical

### 1.1.1 Technical Feasibility

The **Plant Disease Detection system** leverages advanced and widely supported technologies. The machine learning model will be built using TensorFlow or PyTorch, with a convolutional neural network (CNN) trained on labeled plant disease datasets. OpenCV will be utilized for preprocessing images, ensuring input quality and consistency. The system’s backend will be developed using Flask or Django, while React.js will create a responsive, intuitive front end. Hardware requirements are minimal, with a GPU-enabled system required for training, while end-users need only internet-enabled devices like smartphones or desktops. These readily available tools and technologies make the system technically feasible.

### 1.1.2 Operational Feasibility

The **Plant Disease Detection System** is designed to meet the needs of farmers and agricultural experts who may have limited technical expertise. A simple web-based interface allows users to upload plant images and receive real-time disease diagnosis and treatment recommendations. The design ensures ease of use, requiring minimal training to operate the system effectively. The system is scalable, making it accessible for both small-scale and large-scale agricultural operations, with deployment feasible in areas with basic internet access.

### 1.1.3 Economic Feasibility

The **Plant Disease Detection System** is a cost-effective and valuable solution for agriculture. It leverages free tools like Python, TensorFlow, and OpenCV, eliminating software licensing costs. The system can be developed and deployed using existing hardware, such as a standard computer with a GPU for model training, while farmers only require basic smartphones or computers for using the application. This keeps hardware costs reasonable. Once implemented, the system requires minimal maintenance, such as periodic model updates and occasional server management, which are not costly. By automating plant disease identification, the system saves farmers the expense of hiring experts for manual diagnoses and reduces crop losses through timely intervention. These tangible savings offset the initial setup cost over time.

Additionally, the system is designed to scale, allowing more farmers to use it without significant increases in operating costs. It can also integrate with mobile platforms to reach more users in rural areas. Overall, the Plant Disease Detection System is an affordable and sustainable solution that improves agricultural productivity, reduces losses, and supports farmers efficiently.

### 1.1.4 Schedule Feasibility

The **Plant Disease Detection System** can be completed within a realistic and achievable time frame. The development process involves key steps like collecting and preprocessing the dataset, designing and training the machine learning model, developing the user interface, integrating the backend system, and testing the application. By leveraging ready-made tools and libraries such as Python, TensorFlow, and OpenCV, the development process is streamlined, reducing time spent on building components from scratch. With a well-structured plan and a dedicated team, the project can be completed within 3 to 6 months, depending on the complexity of the model and additional features like mobile integration or disease severity estimation. This makes the project schedule both practical and feasible.

### 1.1.5 Specification Feasibility

The **Plant Disease Detection System** meets all the necessary specifications required for effective plant disease identification. It utilizes Python libraries like TensorFlow and OpenCV for machine learning and image processing, which are widely supported and readily available. The system’s specifications, such as uploading plant images, classifying diseases, and providing actionable treatment recommendations, are achievable using current technologies. The software is designed to run efficiently on standard computing devices, with a basic GPU required for model training and standard smartphones or computers sufficient for end-user interaction. These specifications are practical, ensuring the system can be implemented without exceeding resources or technical constraints.

### 1.1.6 Information Feasibility

The information required for the **Plant Disease Detection System** is comprehensive, reliable, and highly applicable to real-world agricultural challenges. The system is designed to leverage well-established tools and technologies, such as TensorFlow and OpenCV, which are supported by extensive documentation and communities, ensuring a smooth development process. The reliability of the system stems from its foundation in machine learning, utilizing labeled datasets of plant diseases to ensure accurate and consistent identification under diverse conditions. The meaningfulness of the information generated is evident in its ability to guide farmers with precise disease diagnoses and actionable treatment recommendations. By addressing issues like crop loss and food security, the system provides essential, timely, and impactful insights, ensuring its practicality and value for end users.

### 1.1.7 Motivational Feasibility

For the **Plant Disease Detection System** to succeed, it is essential that farmers and agricultural experts are motivated to use it consistently and correctly. The system simplifies their work by automating the disease detection process, eliminating the need for manual diagnosis or expert consultations, and providing timely recommendations. Its user-friendly interface ensures that minimal training is required, making it accessible even to users with limited technical expertise. By saving time, reducing crop losses, and offering actionable insights, the system provides clear value, encouraging users to adopt it and follow the necessary steps for effective plant disease management.

### 1.1.8 Legal & Ethical Feasibility

The **Plant Disease Detection System** adheres to all relevant legal and ethical considerations, ensuring compliance with intellectual property rights and data privacy laws. The system uses publicly available or properly licensed datasets for training the machine learning model, avoiding any unauthorized use of proprietary information. From an ethical perspective, the system promotes fairness by being accessible to farmers and agricultural experts regardless of their socioeconomic background. It ensures transparency by clearly communicating its limitations, such as the scope of diseases it can detect, to avoid misleading users. Additionally, the system does not discriminate against specific crops or regions, supporting equitable access to technology for improved agricultural practices. This makes the project legally compliant and ethically sound.

## 1.2 Project/Product Scope

The scope of the Plant Disease Detection System outlines the boundaries of the project, specifying what will be included and excluded. The primary focus is to automate the detection of plant diseases through image analysis powered by machine learning. Key features include uploading plant images, processing them through trained models, identifying diseases, and providing actionable treatment recommendations.

To ensure the project's success, resources such as time, personnel, and budget will be allocated efficiently to develop the essential features within the defined limits. For instance, priority will be given to fundamental functionalities like image preprocessing, disease classification, and a user-friendly interface, while more advanced features like disease severity analysis or multilingual support may be deferred for future iterations if resources are limited.

The scope will be further divided into smaller, manageable components, such as:

* Gathering and preprocessing a dataset of plant disease images.
* Building and training a machine learning model for disease detection.
* Designing an intuitive user interface for image upload and disease feedback.
* Conducting thorough testing and validation to ensure accuracy and reliability.

This well-defined scope will guide the development of a robust and effective Plant Disease Detection System tailored to the needs of farmers and agricultural professionals.

## 1.3 Project/Product Costing

To estimate the cost of developing the **Plant Disease Detection System**, we will use metrics to evaluate and measure different aspects of the project. These metrics will guide us in tracking progress and ensuring the quality of the final product. There are two main types of metrics:

1. **Knowledge-Oriented Metrics:** These metrics will track the progress of key tasks, such as:

* Collecting and preprocessing the plant disease image dataset.
* Training and fine-tuning the machine learning model for disease classification.
* Designing and implementing the user interface for farmers and agricultural experts.
* Testing and validating the system to ensure reliability.

These metrics will help monitor whether the project is on schedule and predict the resources needed at different stages of development.

2. **Achievement-Oriented Metrics:** These metrics will measure the quality of the final system, such as:

* The accuracy of the disease detection model (e.g., achieving at least 80% classification accuracy).
* The responsiveness and ease of use of the user interface.
* The system’s ability to provide clear and actionable disease treatment recommendations.

These metrics will ensure the system meets its performance goals and user requirements.

To estimate costs, we will use a **cost model** that links required resources (time, personnel, and computational tools) to these metrics. For instance:

* Time and labor will include tasks such as dataset preparation, model training, and UI/UX development.
* Equipment and tools will involve high-performance hardware (e.g., GPUs for training the model) and software frameworks like TensorFlow, Flask, and OpenCV.

This estimation will be informed by historical data from similar machine learning and software projects and adjusted to reflect specific needs, such as model complexity, dataset size, and performance targets identified in the project proposal. By aligning the cost estimation with the goals and tasks in the proposal, we can ensure accurate budgeting and effective resource allocation for the Plant Disease Detection System.

### 1.3.1 Project Cost Estimation By Function Point Analysis

**Function Point Analysis (FPA)** is a method to estimate the cost and effort of a software project by analyzing its functional requirements. To apply FPA to the **Plant Disease Detection System**, we break the project into measurable components and assign weights based on their complexity. Here’s how we can implement it step-by-step:

**FPA Factors for the Plant Disease Detection System:**

By breaking down the project into five functional components typically analyzed in FPA:

**1.External Inputs:** These are inputs provided by users to the system.

* **Image Upload:** Users upload plant images for disease detection.
* **User Registration:** Users provide their details for creating accounts.
* **Feedback Submission:** Users submit feedback or requests for updates to disease databases.
* **Entering crop details** (e.g., plant type, location, growth stage).

**2. External Outputs:** These are the outputs the system provides to users.

* **Disease Identification Results:** Provides the disease name identified from the uploaded image.
* **Treatment Recommendations:** Suggests actionable steps for managing the identified disease.
* **Generating reports** (e.g., downloadable diagnosis summary).

**3. External Inquiries:** These are specific user-driven queries to retrieve data.

* **Query Disease History:** Users can inquire about their previous disease detection results.
* **Search for Disease Information:** Users can look up details about specific plant diseases in the system.
* **Checking system usage or performance stats** (optional, for advanced systems).

**4. Internal logical Files:** Logical files or data stores the system uses to manage information.

* **Labeled Dataset:** Contains disease-specific labeled images for training and testing.
* **Trained Model File:** Stores the machine learning model used for disease detection.
* **System Logs:** Maintains logs for system operations and user activities.
* **Treatment Recommendations Data** (e.g., remedies linked to specific diseases).

5. **External Interfaces Files:** Interfaces with other systems or applications for data exchange.

* **Mobile App Integration:** For extending disease detection to mobile platforms.
* **Third-Party APIs:** For retrieving additional agricultural or disease data.
* **Weather Data API** (e.g., to correlate disease occurrence with environmental conditions).
* **Agriculture Database** (e.g., external datasets for plant diseases or soil conditions).

Table for Unadjusted Function Points:

|  |  |  |  |
| --- | --- | --- | --- |
| Function Type | Low | Medium | High |
| EI | 3 | 4 | 6 |
| EO | 4 | 5 | 7 |
| EQ | 3 | 4 | 6 |
| ILF | 7 | 10 | 15 |
| EIF | 5 | 6 | 10 |

**Calculating Unadjusted Function Points:**

Calculate unadjusted function points by using the formulae:

**UFP = =∑( Count×Weight)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| UFP Factor | Count | Complexity | Weight | Total |
| EI | 4 | Low | 3 | 12 |
| EO | 3 | Low | 4 | 12 |
| EQ | 3 | Low | 3 | 09 |
| ILF | 4 | Medium | 10 | 40 |
| EIF | 4 | Medium | 7 | 28 |
| Total UFP |  |  |  | 101 |

**Value Adjustment Factors**

Value Adjustment Factors (VAF) are used to adjust the Unadjusted Function Points (UFP) based on system characteristics like performance, usability, and maintainability, reflecting the system's complexity.

**Formulae for Calculating VAF**

VAF = 0.65+(0.01\*Sum of GSCs)

Where **General System Characteristics (GSCs)** in the Value Adjustment Factor (VAF) are 14 factors used to assess the impact of system-specific attributes like performance, security, and portability on the overall software development effort. These Factors rated from 0 (no influence) to 5 (strong influence).

|  |  |
| --- | --- |
| GSCs | Rating |
| Data communications | 3 |
| Distributed data processing | 3 |
| Performance | 4 |
| Heavily used Configuration | 3 |
| Transaction Rate | 2 |
| On-Line Data Entry | 4 |
| End User Efficiency | 4 |
| On-Line Update | 3 |
| Complex Processing | 4 |
| Reusability | 3 |
| Installation Ease | 3 |
| Operational Ease | 4 |
| Multiple Sites | 2 |
| Facilitate Change | 3 |
| Total | 45 |

Total Degree of influence = 45

By using the Formulae and putting the value

VAF= 0.65+(0.01\*45)

|  |
| --- |
| **VAF=1.1** |

**Function Point Analysis**

FP=UFP \*VAF

FP=101 \*1.1

**FP=111.1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type of Components** | **Complexity of Components** | | | |
|  | Low | Average | High | Total |
| External Input | 3 | 4 | 6 | 12 |
| External Output | 4 | 5 | 7 | 12 |
| External Inquiry | 3 | 4 | 6 | 09 |
| Internal Logical Files | 7 | 10 | 15 | 40 |
| External Interfaces Files | 5 | 7 | 10 | 28 |
| **Total Number of UFP** | | | **101** |
| **Total Number of VAF** | | | **1.1** |
| **Total FPA** | | | **111.1** |

Finally, Total Project Cost and Total Project Effort are calculated given the average productivity parameter for the system.

The formulae are given as follows:

**Cost / FP = labor rate / productivity parameter**

**Total Project Cost = FP est. \* (cost / FP)**

**Total Estimated Effort = FP est. / productivity parameter**

By using these formulas, we will calculate the total project cost and total project estimate effort:

Average Labor Rate = $4/Hour

Productivity Parameter = 05FP/Hour

|  |  |
| --- | --- |
| Description | Values |
| FP Estimate | 111.1 |
| Labor Rate | $4/Hour |
| Productivity Parameter | 5FP/Hour |
| Cost per FP | 0.8$/FP |
| Total Project Cost | 88.8$ |
| Total Estimated effort | 22.22 hrs. |

### 1.3.2 Project Cost Estimation by using COCOMO’81 (Constructive Cost Model)

Boehm's COCOMO model is one of the mostly used models commercially. The first version of the model delivered in 1981 and COCOMO II is available now. COCOMO 81 is a model that allows one to estimate the cost, effort, and schedule when planning a new software development activity, according to software development practices that were commonly used in the 1970s through the 1980s. It exists in three forms, each one offering greater detail and accuracy the further along one is in the project planning and design process. Listed by increasing fidelity, these forms are called Basic, Intermediate, and Detailed COCOMO. However, only the Intermediate form has been implemented by USC in a calibrated software tool.

Three levels:

**Basic:** Is used mostly for rough, early estimates.

**Intermediate:** Is the most commonly used version, includes 15 different factors to account for the influence of various project attributes such as personnel capability, use of modern tools, hardware constraints, and so forth.

**Detailed:** Accounts for the influence of the different factors on individual project phases: design, coding/testing, and integration/testing. Detailed COCOMO is not used very often.

Each level includes three software development types:

1. **Organic:** Relatively small software teams develop familiar types of software in an in-house environment. Most of the personnel have experience working with related systems.
2. **Embedded:** The project may require new technology, unfamiliar algorithms, or an innovative new method
3. **Semi-detached:** Is an intermediate stage between organic and embedded types.

Implementing Constructive Cost Model

1. Determining the Project Type:

COCOMO has three basic types of projects such as Organic, Semi-detached, and Embedded.

The Plant Disease Detection System is Semi-Detached because

1. Moderate Complexity: Combines machine learning, UI/UX, and system integration.
2. Partial Team Familiarity: Mix of known and unfamiliar technologies/techniques.
3. Balance of Routine and Research: Involves standard development and innovative experimentation.
4. Effort and Development Time Formulas:

* Efforts (PM):

Effort refers to the total amount of work needed for the project, measured in person-months (PM).

E = a×(KLOC)b

Where:

a and b are constants determined based on the project type and scale

KLOC is the number of thousands of lines of code (thousands of lines of code).

* Development Time (T):

Development time is the time needed to complete the project, typically measured in months.

T = c×(E)d

Where:

c and d are constants

E is the effort from the previous formula.

* Team Size (S)

Team size is the number of people working on the project, usually calculated as:

S = E/T

Where:

E is the total effort.

T is the development time.

For Organic Projects

a = 3.0, b = 1.12

c = 2.5, d = 0.35

1. Estimating KLOC:
2. Dataset Collection and Pre-Processing: 1KLOC (using Python libraries like PyTorch and OpenCV).
3. Model Development and Training: 2KLOC (depends on model complexity)
4. Backend Development: 3KLOC (Integration and API Logic)
5. Frontend Development: 2KLOC (User Friendly Design)
6. Database Development and Integration: 1KLOC (compact size due to simple requirements)

Total Estimated KLOC: 09 KLOC

1. Calculating Effort:

|  |  |  |
| --- | --- | --- |
| a | KLOC | b |
| 3.0 | 09 | 1.12 |

By Using Effort Estimation Formula

E = 3.0\*(09)^1.12

E = 3.0\*11.71

E = 35.13PM

1. Calculating Development Time:

|  |  |  |
| --- | --- | --- |
| c | Efforts | d |
| 2.5 | 35.13 | 0.35 |

By Using the Formula

D = 2.5\*(35.13)^0.35

D = 2.5\*3.47

D = 8.67(Months)

1. Calculate Team Size:

|  |  |
| --- | --- |
| E | T |
| 35.13 | 8.67 |

By using the Formula

S = 35.13/8.67

S = 4.05 People

|  |  |
| --- | --- |
| Effort (Person – Months) | 35.13 |
| Development Time | 8.67 (Months) |
| Team Size Required | 4.05 people |

## 1.4 CPM - Critical Path Method

In 1957, DuPont developed a project management method designed to address the challenge of shutting down chemical plants for maintenance and then restarting the plants once the maintenance had been completed. Given the complexity of the process, they developed the Critical Path Method (CPM) for managing such projects.

CPM provides the following benefits:

* Provides a graphical view of the project.
* Predicts the time required to complete the project.
* Shows which activities are critical to maintaining the schedule and which are not.

CPM models the activities and events of a project as a network. Activities are depicted as nodes on the network and events that signify the beginning or ending of activities are depicted as arcs or lines between the nodes. The following is an example of a CPM network diagram:

Steps in CPM Project Planning

1. Specify the individual activities.

2. Determine the sequence of those activities.

3. Draw a network diagram.

4. Estimate the completion time for each activity.

5. Identify the critical path (longest path through the network)

6. Update the CPM diagram as the project progresses.

**1. Specify the Individual Activities**

By breaking down the project into smaller and more manageable parts, the key activities are:

1. Project Initialization & Planning
2. Dataset Collection and Preprocessing
3. Model Development
4. Model Training and Evaluation
5. Backend Development
6. Frontend Development
7. Integration of Components
8. Testing and Validation
9. Deployment

**2. Determine the Sequence of the Activities**

Dependencies between the activities are:

* **Project Initialization & Planning** → Must precede all other activities.
* **Dataset Collection and Preprocessing** → Must be completed before Model Development and Training.
* **Model Development** → Must precede Model Training and Evaluation.
* **Model Training and Evaluation** → Must be completed before Integration of Components.
* **Backend and Frontend Development** → Can proceed in parallel but must finish before Integration of Components.
* **Integration of Components** → Depends on completion of Backend, Frontend, and Model Training.
* **Testing and Validation** → Follows Integration of Components.
* **Deployment** → Final step after successful Testing and Validation.

**3. Draw the Network Diagram**

The network diagram would include nodes (activities) and arrows (dependencies). Here’s the structure in order:

* **Start → Project Initialization & Planning → Dataset Collection and Preprocessing → Model Development → Model Training and Evaluation → Integration of Components → Testing and Validation → Deployment → End**

Backend and Frontend Development will run parallelly after Dataset Collection and Preprocessing and merge at Integration.

B

A

F

E

C

I

H

G

D

**Figure 1 Network Diagram of Plant Disease Detection System**

**4. Estimate Activity Completion Time**

|  |  |
| --- | --- |
| Activities | Completion Time (Weeks) |
| Project Initialization & Planning | 2 Weeks |
| Dataset Collection and Preprocessing | 4 Weeks |
| Model Development | 6 Weeks |
| Model Training & Evaluation | 8 Weeks |
| Backend Development | 6 Weeks |
| Frontend Development | 5 Weeks |
| Integration of Components | 3 Weeks |
| Testing and Validation | 4 Weeks |
| Deployment | 2 Weeks |

**5. Identify the Critical Path**

The critical path is the longest-duration path through the network. The significance of the critical path is that the activities that lie on it cannot be delayed without delaying the project. Because of its impact on the entire project, critical path analysis is an important aspect of project planning.

Determining the following six parameters for each activity which can identify the critical path:

**ES:** earliest start time: the earliest time at which the activity can start given that its precedent activities must be completed first.

ES (K)= max [EF(J) : J is an immediate predecessor of K]

**EF:** earliest finish time: equal to the earliest start time for the activity plus the time required to complete the activity.

EF (K)= ES (K) + Dur (K)

**LF:** latest finish time: the latest time at which the activity can be completed without delaying the project.

LF (K)= min [LS(J) : J is a successor of K]

**LS:** latest start time: equal to the latest finish time minus the time required to complete the activity.

LS (K)= LF(K) – Dur (K)

**TS:** Total Slack: the time that the completion of an activity can be delayed without delaying the end of the project

TS (K)= LS(K) – ES(K)

**FS:** Free Slack: the time that an activity can be delayed without delaying both the start of any succeeding activity and the end of the project.

FS (K)= min [ES(J) : J is successor of K] – EF(K)

The slack time for an activity is the time between its earliest and latest start time, or between its earliest and latest finish time. Slack is the amount of time that an activity can be delayed past its earliest start or earliest finish without delaying the project.

The critical path is the path through the project network in which none of the activities have slack, that is, the path for which ES=LS and EF=LF for all activities in the path. A delay in the critical path delays the project. Similarly, to accelerate the project it is necessary to reduce the total time required for the activities in the critical path.

**Critical Path for Plant Disease Detection System:**

We will first calculate all the parameters step by step:

|  |  |  |  |
| --- | --- | --- | --- |
| Task ID | Activities | Duration (Weeks) | Predecessors |
| A | Project Initialization & Planning | 2 Weeks | None |
| B | Dataset Collection and Preprocessing | 4 Weeks | Planning |
| C | Model Development | 6 Weeks | Dataset Pre-Processing |
| D | Model Training & Evaluation | 8 Weeks | Model Development |
| E | Backend Development | 6 Weeks | Dataset Pre-Processing |
| F | Frontend Development | 5 Weeks | Dataset Pre-Processing |
| G | Integration of Components | 3 Weeks | Model Training, Backend, Frontend |
| H | Testing and Validation | 4 Weeks | Integration |
| I | Deployment | 2 Weeks | Testing |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Activity** | **Duration**  **(Weeks)** | **Predecessor(s)** | **ES** | **EF** | **LS** | **LF** | **TS** | **FS** |
| Project Initialization & Planning | 2 | None | 0 | 2 | 0 | 2 | 0 | 0 |
| Dataset Collection and Preprocessing | 4 | Planning | 2 | 6 | 2 | 6 | 0 | 0 |
| Model Development | 6 | Dataset Pre-Processing | 6 | 12 | 6 | 12 | 0 | 0 |
| Model Training & Evaluation | 8 | Model Development | 12 | 20 | 12 | 20 | 0 | 0 |
| Backend Development | 6 | Dataset Pre-Processing | 6 | 12 | 12 | 18 | 6 | 6 |
| Frontend Development | 5 | Dataset Pre-Processing | 6 | 11 | 15 | 20 | 9 | 9 |
| Integration of Components | 3 | Model Training, Backend, Frontend | 20 | 23 | 20 | 23 | 0 | 0 |
| Testing and Validation | 4 | Integration | 23 | 27 | 23 | 27 | 0 | 0 |
| Deployment | 2 | Testing | 27 | 29 | 27 | 29 | 0 | 0 |

**So based on calculation of these Parameters,**

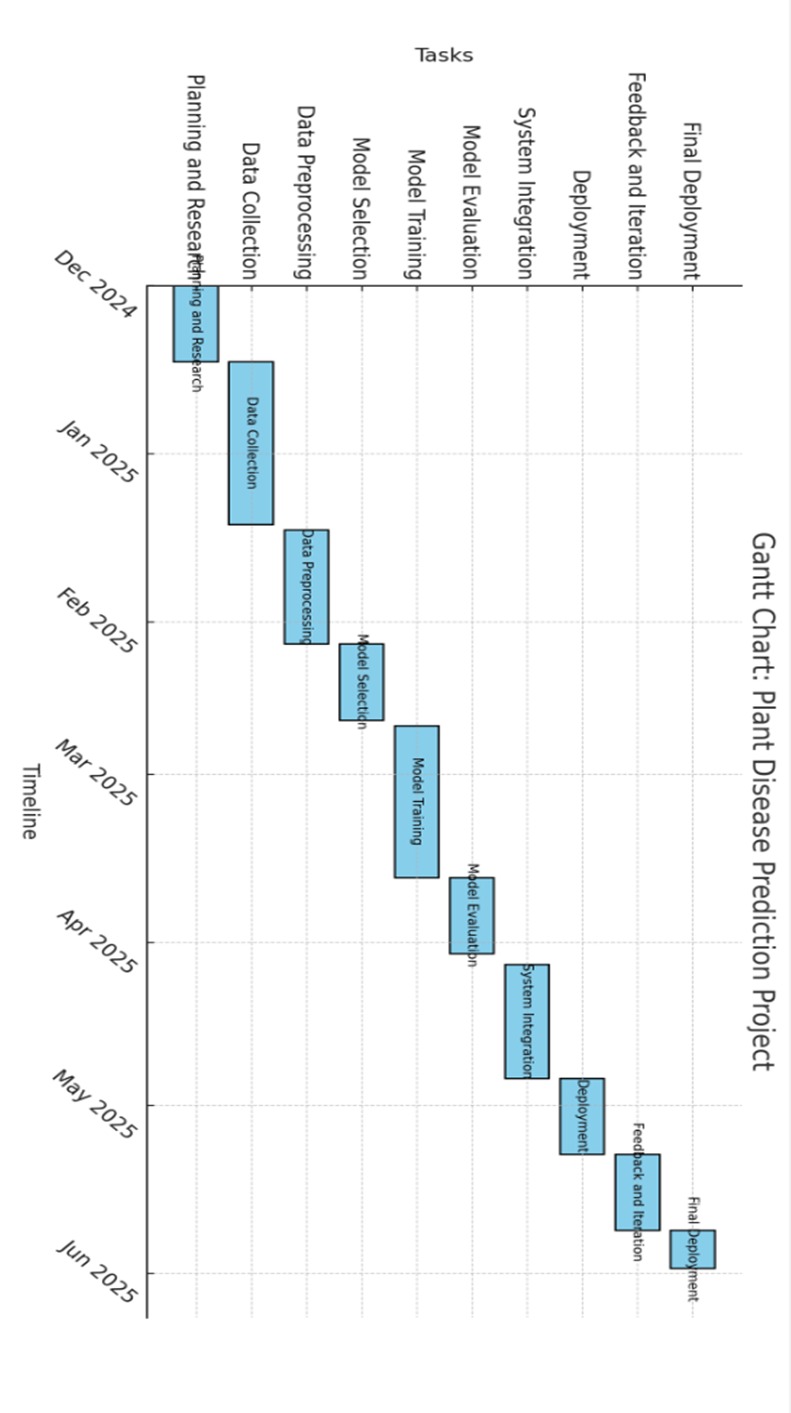
**The Critical Path is :**

**A → B → C → D → G → H → I (29 Weeks)**

**6. Update CPM Diagram**

As the project progresses, the actual task completion times will be known and the network diagram can be updated to include this information. A new critical path may emerge, and structural changes may be made in the network if project requirements change.

## 1.5 Gantt chart



## 1.6 Introduction to Team member and their skill set

**Team Members and Task Assignments**

**Muhammad Tayyab**

**Role:** Leading Developer

**Skills:** Got Excellent Skills in Python Programming for Backend development and database management. Experienced in project planning and implementing systems using OpenCV, PyTorch and TensorFlow.

**Assigned Tasks**

1. Requirement Gathering
2. UI/UX Design
3. Backend Development
4. Database Integration and Management
5. Deployment

**Muhammad Sohaib Raza**

**Role:** Supportive Developer

**Skills:** Got intermediate skills in Python and Javascript Programming and UI/UX Design. Willing to learn more and assist team members in different tasks.

**Assigned Tasks**

1. Frontend Development
2. UI/UX Design (Assisting in Creation of User Interface)
3. Model Development and Training Assistance
4. Project Handover Documentation
5. Component Testing

**Mirza Shazil Arsal**

**Role:** Model Developer

Skills: Highly Skilled in Machine Learning Using CNN (Keras) to develop and train model for testing. Got intermediate knowledge of Python Programming to assist in Backend Development using OpenCV.

**Assigned Tasks**

1. Model Development and Training
2. Backend Development in Assistance
3. Deployment Assistance
4. Testing (Model’s accuracy and Usability)

## 1.7 Tools and Technology with Reasoning

The following tools and technologies will be used for the development of the Plant Disease Detection System, ensuring effective implementation and alignment within the project's requirements, constraints, and goals:

1. **Frontend Tools**

**React.js**

* + **Reason:** React.js provides a modern, component-based approach for building dynamic and responsive web interfaces. It allows seamless integration with REST APIs for uploading plant images and displaying disease results.
  + **Needs Addressed:** A user-friendly and responsive interface for farmers to interact with the system, upload plant images, and view results.
  + **Constraints:** React.js may require additional setup for state management (e.g., Redux) if the application scales significantly.

**Tailwind CSS**

* + **Reason:** Tailwind CSS enables rapid design and customization of interfaces with prebuilt utility classes.
  + **Needs Addressed:** Simplifies the creation of visually appealing and consistent UI components.
  + **Constraints:** Requires familiarity with utility-based CSS and integration into React.js projects.

1. **Backend Tools**

**Flask or Django (Python)**

* + **Reason:** Both frameworks are lightweight, scalable, and integrate seamlessly with machine learning models. Flask is ideal for smaller applications, while Django supports rapid development with its built-in admin panel.
  + **Needs Addressed:** Creating REST APIs to handle image uploads, process disease detection using ML models, and deliver results to the frontend.
  + **Constraints:** Requires proper error handling and optimization for concurrent requests.

**Python**

* + **Reason:** Python is versatile, with extensive libraries for machine learning (e.g., TensorFlow, PyTorch) and image processing (e.g., OpenCV).
  + **Needs Addressed:** Core functionality, including ML model integration and disease classification.
  + **Constraints:** Dynamic typing may lead to runtime errors, requiring robust testing and debugging.

1. **Database Tools**

**PostgreSQL**

* + **Reason:** A powerful relational database system with support for complex queries and data types.
  + **Needs Addressed:** Efficient storage of user data, image metadata, and logs.
  + **Constraints:** Requires schema design optimization for scalability and performance.

**SQLite (During Development)**

* + **Reason:** A lightweight, file-based database for rapid prototyping and testing.
  + **Needs Addressed:** Initial storage and testing before migrating to PostgreSQL for production.
  + **Constraints:** Limited scalability compared to PostgreSQL.

1. **Machine Learning and Image Processing Tools**

**TensorFlow or PyTorch**

* + **Reason:** Both frameworks are industry standards for developing, training, and deploying machine learning models.
  + **Needs Addressed:** Building and deploying convolutional neural networks (CNNs) for disease detection.
  + **Constraints:** Training models can be computationally expensive without GPU support.

**OpenCV**

* + **Reason:** OpenCV is a robust library for image preprocessing (e.g., resizing, augmentation, normalization).
  + **Needs Addressed:** Preparing plant images for consistent input into ML models.
  + **Constraints:** May require tuning for specific image quality issues (e.g., blurry or noisy images).

1. **Development Platforms**

**Google Colab**

* + **Reason:** Provides free access to GPUs/TPUs for model training and prototyping.
  + **Needs Addressed:** Accelerates ML model development and testing.
  + **Constraints:** Limited resources compared to paid cloud solutions.

**Local Machine**

* + **Reason:** Develop and test backend and frontend components on personal computers.
  + **Needs Addressed:** Offline development environment.
  + **Constraints:** May require additional setup for dependencies (e.g., Python, Node.js).

1. **Testing Tools**

**Pytest**

* + **Reason:** A powerful testing framework for Python applications.
  + **Needs Addressed:** Unit testing for backend and ML components.
  + **Constraints:** Requires comprehensive test cases for effective validation.

**Postman**

* + **Reason:** Simplifies API testing by enabling quick request/response validations.
  + **Needs Addressed:** Ensures API endpoints function correctly.
  + **Constraints:** Requires manual testing unless integrated with automated workflows.

1. **Version Control**

**GitHub**

* + **Reason:** Enables version control, collaboration, and deployment workflows.
  + **Needs Addressed:** Facilitates code sharing and team collaboration.
  + **Constraints:** Requires team familiarity with Git commands and workflows.

1. **Deployment Tools**

**Docker**

* + **Reason:** Containerizes the application for consistent deployment across environments.
  + **Needs Addressed:** Ensures seamless deployment of ML models, backend, and frontend.
  + **Constraints:** Requires knowledge of containerization and Dockerfiles.

**Heroku or AWS**

* + **Reason:** Cloud platforms for deploying the application with scalability options.
  + **Needs Addressed:** Hosting the application and serving users globally.
  + **Constraints:** Costs can increase with scale and traffic.

1. **Budget and Time Constraints**

**Budget-Friendly Tools**

The project primarily uses open-source tools and technologies to minimize costs, while still maintaining functionality and scalability. These tools include Python, PostgreSQL React.js and Tailwind CSS, TensorFlow/PyTorch OpenCVand Google Colab.

**Time Efficiency**

To ensure the project is completed within the planned timeline of **29 weeks**, we utilize pre-built libraries and frameworks to speed up development:

* **Pre-Built Libraries**:

TensorFlow/PyTorch

OpenCV

## 1.8 Vision Document

The **Plant Disease Detection System** is designed to automate the process of identifying plant diseases using machine learning (ML) techniques. The system aims to assist farmers, agricultural experts, and other stakeholders by providing timely and accurate disease diagnoses through a user-friendly interface. This solution addresses the challenges of manual inspection, which is often time-consuming, error-prone, and heavily reliant on expert availability.

**1.** **Stakeholders**

**Farmers**:

* Benefit from an accessible tool that provides disease detection and treatment recommendations.
* Reduce dependency on experts for disease identification.

**Agricultural Experts**:

* Use the system as a supportive tool to confirm diagnoses and recommend treatments.

**Agricultural Organizations and NGOs**:

* Leverage the system to improve crop health and support farmers in rural areas.

**IT Administrators**:

* Maintain system performance, update models, and manage data securely.

**2. System Objectives**

**Automation**:

* Automate disease identification by analyzing plant images.

**Accuracy**:

* Provide highly accurate results by leveraging trained machine learning models.

**Usability**:

* Develop an easy-to-use interface for non-technical users like farmers.

**Efficiency**:

* Deliver results in real-time to minimize delays in decision-making.

**Scalability**:

* Ensure the system can adapt to additional plant species and diseases.

**3. Core Features**

**Image Upload**:

* Users upload images of diseased plants via the system interface.

**Disease Detection**:

* The system identifies the disease using a trained ML model.

**Treatment Recommendations**:

* Provide detailed suggestions to treat identified diseases.

**Database Management**:

* Store user data, image metadata, and logs for future reference and analysis.

**Reporting**:

* Generate insights on the system’s accuracy and usage trends.

**4. System Boundaries**

**Included**:

* Disease detection for predefined diseases (e.g., fungal, bacterial, viral).
* Treatment recommendations for identified diseases.
* User-friendly web-based interface.

**Excluded**:

* Real-time field analysis of environmental conditions.
* Support for unidentified or emerging diseases.
* Mobile application development (considered for future phases).

**5. Assumptions**

* The dataset used for training the ML model contains sufficient labeled images of diseases.
* The users have access to devices (e.g., smartphones, computers) with internet connectivity.
* Images uploaded to the system are of good quality and resolution.

**6. Constraints**

**Technical**:

* Limited to diseases included in the training dataset.
* Requires sufficient computational power for ML model training.

**Economic**:

* The project depends on open-source tools to minimize costs.

**Time**:

* Must be completed within the allocated 29-week schedule.

**Resource**:

* A team of three developers working on ML, frontend, and backend components.

**7. Checkpoints**

**Problem Statement**:

* Automate disease detection using ML to improve crop health and reduce losses.

**Stakeholders**:

* Farmers, agricultural experts, organizations, and IT administrators.

**System Boundaries**:

* Focuses on disease detection, classification, and treatment suggestions.

**Constraints**:

* Limited by dataset size, computational power, and development resources.

**Features Validation**:

* Ensures high accuracy and usability to meet stakeholder needs.

**Conclusion**

The **Plant Disease Detection System** will provide a cost-effective, efficient, and accurate solution to the problem of plant disease identification. By automating the process and making it accessible through a web-based platform, the system empowers farmers and agricultural experts to make timely decisions, ultimately enhancing crop health and productivity. This project aligns with the broader goal of addressing global agricultural challenges and ensuring food security.

## 1.9 Risk List

|  |  |  |  |
| --- | --- | --- | --- |
| Risk | Likelihood | Impact | Mitigation |
| **Poor Dataset Quality** | High | High | Source high-quality, diverse datasets from trusted repositories. Perform preprocessing (e.g., augmentation, noise removal). |
| **Insufficient Dataset Size** | Medium | High | Use data augmentation techniques. Leverage transfer learning with pretrained models. Collaborate with agricultural experts for more data. |
| **Overfitting of the Model** | Medium | High | Use regularization techniques (e.g., dropout, weight decay). Ensure dataset has a balanced train-test split. Validate with unseen data. |
| **Difficulty in Model Integration** | Medium | Medium | Use standardized frameworks (e.g., TensorFlow, Flask). Allocate extra time for integration. Test API functionality early. |
| **Inconsistent Model Predictions** | Medium | High | Perform extensive testing on multiple datasets. Add confidence scores to predictions. Regularly retrain the model with updated data. |
| **Backend-Frontend Integration Issues** | Medium | High | Use APIs to standardize communication. Conduct integration tests after completing backend and frontend modules. Assign dedicated time for debugging. |
| **Time Overruns** | Medium | High | Create a detailed project timeline using tools like Gantt charts. - Set intermediate milestones and review progress weekly. |
| **Limited GPU/Hardware Resources** | Low | High | Use cloud-based resources (e.g., Google Colab, AWS) for model training. Optimize the model for lightweight inference. |
| **Security Vulnerabilities** | Low | Medium | Secure APIs using authentication and encryption. Regularly review code for vulnerabilities. Follow security best practices for deployment. |

**Risk Management Process**

* **Identification:** Risks are identified at the start of the project and throughout its lifecycle.
* **Assessment:** Each risk is evaluated based on likelihood and potential impact.
* **Mitigation:** Plans are developed to reduce the likelihood or impact of risks.
* **Monitoring:** Risks are reviewed at the end of each iteration, and the list is updated as necessary.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*REQUIREMENTS ENGINEERING\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## 

## 1 Introduction

Requirements engineering process provides the appropriate mechanism for understanding what the customer wants, analyzing need, assessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the specification and managing the requirements as they are transformed into an operational system. The task of capturing, structuring, and accurately representing the user's requirements so that they can be correctly embodied in systems which meet those requirements (i.e. are of good quality).

* Requirements elicitation
* Requirements analysis and negotiation
* Requirements specification
* System modeling
* Requirements validation
* Requirements management

### Requirement Elicitation

**1. Identify Stakeholders:**

Elicitate Requirements from different Stakeholders such as:

**Primary Users:** (Farmers, agricultural experts, researchers).

**Development Team**.

**Domain Experts**: Agronomists, plant pathologists.

**Other Stakeholders**: Academic supervisors, IT administrators.

**2. Techniques for Requirement Elicitation**

**a. Document Analysis:** Analyze your project proposal and existing research papers on plant disease detection systems.

**b. Interviews:** Conduct structured or semi-structured interviews with stakeholders.

**c. Surveys/Questionnaires:** Distribute surveys to potential users (e.g., farmers, researchers).

Collect a broad range of input on user expectations, such as interface simplicity or the desired output format.

**d. Observation:** Observe real-world scenarios in which farmers deal with diseased plants.

**E. Focus Groups:** Facilitate discussions among a group of stakeholders (e.g., farmers and agricultural experts).

### Requirement Analysis and Negotiation

This involves evaluating the elicited requirements to ensure they are valid, clear, and feasible.

**1. Categorize Requirements**

**Functional Requirements**: What the system should do. (e.g: The system must detect diseases from plant images).

**Non-Functional Requirements**: System performance and quality attributes. (e.g: The system must deliver results within 5 seconds.)

**Constraints**: Technical, budgetary, or timeline limitations. (e.g: The model should be trained using publicly available datasets).

**2. Validate Requirements**

**Check for completeness:** Ensure no key features or user needs are missing.

**Check for clarity:** Avoid ambiguous requirements. Example: Instead of "The system must be fast," say "The system must deliver results within 5 seconds."

**Check for feasibility:** Ensure requirements are realistic given the time, resources, and skills available.

**3. Identify Conflicts**

Look for conflicting requirements (e.g., high accuracy vs. limited dataset).

**4. Organize a Stakeholder Meeting**

Include farmers, domain experts, supervisors, and your team.

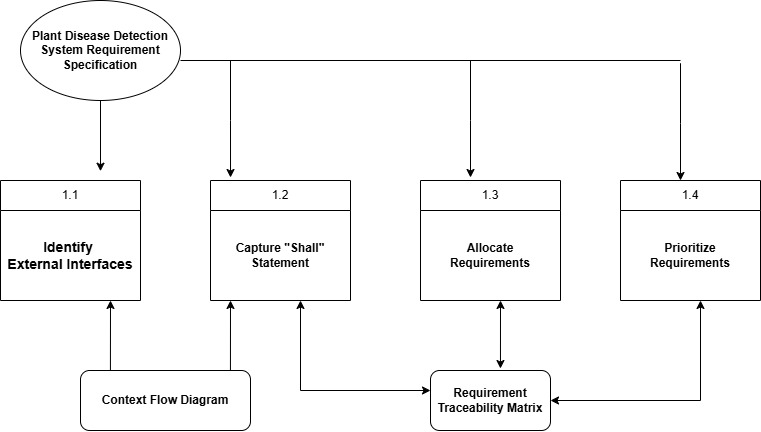
Present the analyzed requirements to them.

**5. Discuss Conflicts and Feasibility**

Explain challenges related to conflicting requirements.

**6. Seek Compromises**

Negotiate trade-offs to resolve conflicts.



Here, requirements specification is to be discussed. Requirements specification would lead to the following four steps:

* Identify external interfaces
* Development of context diagram
* Capture “shall statements
* Allocate requirements
* Prioritize requirements
* Development of requirements traceability matrix

### 1.1 Systems Specifications

**Introduction**

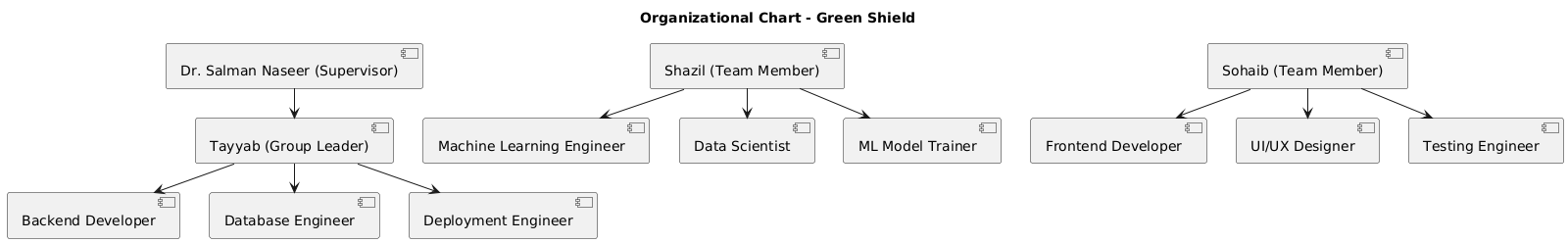
The **Plant Disease Detection System** is an advanced machine learning-based solution designed to assist farmers and agricultural professionals in identifying plant diseases through image analysis. By leveraging deep learning techniques, the system aims to improve agricultural productivity by enabling early detection and treatment recommendations.

The implementation of this system will help reduce crop losses, minimize the use of unnecessary pesticides, and support sustainable farming practices. The system is designed to be user-friendly, allowing farmers to simply upload an image of a plant leaf, after which the system will process the image and provide a disease classification along with potential treatment solutions.

**Existing System**

Currently, farmers and agricultural experts rely on manual inspection methods to identify plant diseases. These traditional methods are prone to human error, require domain expertise, and often lead to delayed responses, resulting in increased crop damage. Moreover, many existing disease detection systems require expensive tools and laboratory testing, making them inaccessible to small-scale farmers.

**Organizational Chart**



**Scope of the System**

**Plant Disease Detection System** will be implemented in four phases phases to ensure efficient progress, adaptability, and long-term sustainability in real-world agricultural applications.

**Phase I**

This phase focuses on Core System Development including:

* Data Collection & Preprocessing
* Machine Learning Model Training
* Basic Web Application
* Database Setup
* Initial Testing

**Phase II**

This Phase will focuses on System Enhancements including:

* Advanced Model Optimization
* User Interface Improvements
* Multilingual Support
* Basic Recommendation System
* Beta Testing with Users

**Phase III:**

Phase III focuses on Feature Expansion which includes:

* Mobile App Development:
* Severity Estimation
* Historical Data Analysis
* Environmental Factor Integration
* Final Testing & Deployment

**Phase IV**

This phase will focus on Future Enhancements & Scalability including:

* AI-based Treatment Recommendations
* Community & Expert Collaboration
* Expansion to More Crops & Diseases
* Performance Optimization

This document primarily covers **Phase-I** , focusing on establishes the project's foundation, ensuring technical feasibility, defining system scope, estimating costs, and setting up critical tools.

**Summary of Requirements (Initial Requirements)**

The proposed **Plant Disease Detection System** must fulfill the following initial requirements:

**Image-Based Disease Detection**

The system must analyze plant images and classify diseases using machine learning algorithms.

**User-Friendly Interface**

Users should be able to upload plant images easily and receive immediate feedback.

**Machine Learning Model Integration**

A trained CNN model must classify plant diseases accurately based on pre-labeled datasets.

**Database Management**

The system must store uploaded images, classification results, and historical disease data for analysis.

**Real-Time Processing**

Image analysis should be performed quickly, and results should be displayed without delay.

**Disease Identification & Recommendations**

The system should identify the disease and provide general treatment suggestions to assist farmers in decision-making.

**Scalability**

The system should be designed to support future expansions, such as adding more crop species and diseases.

**Security & Data Protection**

User data, including uploaded images and classification results, should be encrypted and securely stored.

**Error Handling & Logging**

The system must detect and log errors, such as poor-quality image uploads or failed disease classifications, for troubleshooting and improvements.

These initial requirements focus on automating plant disease detection, ensuring accuracy, and providing actionable insights to assist users in effective crop management.

### 1.2 Identifying External Entities

The identification of the external entities will be based on the information contained in your Abstract.

The Identification of External Entities is done in two phases.

**a. Over Specify Entities from Abstract**

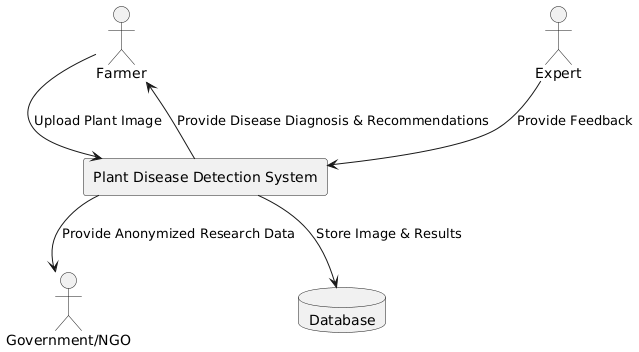
* Farmers
* Agricultural Experts
* Plant Disease Dataset
* Machine Learning Model (CNN / Deep Learning)
* Cloud Storage or Database
* User Interface (Web Application)
* Treatment Recommendation System
* Hardware (Computing device with GPU)
* Internet Connectivity
* Government / NGOs / Agribusinesses
* Treatment Recommendation System

**b. Perform Refinement**

After over specifying the entities, you have to refine them on the basis of your business logic.

* Farmers
* Agricultural Experts
* User Interface (Web Application)
* Treatment Recommendation System
* Government / NGOs / Agribusinesses

### Context Level Data Flow Diagram



### Capture "shall" Statement

|  |  |  |
| --- | --- | --- |
| **ID** | **External Entities** | **Initial Requirements** |
| PDS-01 | Farmer | |  | | --- | | The system **shall** allow users to upload images of plant leaves. |  |  | | --- | |  | |
| PDS-02 | System | The system **shall** process the uploaded images to detect plant diseases using machine learning. |
| PDS-03 | System | |  | | --- | | The system **shall** classify plant diseases based on pre-trained models. |  |  | | --- | |  | |
| PDS-04 | System | The system **shall** provide users with disease diagnosis results. |
| PDS-05 | System | |  | | --- | | The system **shall** suggest treatment recommendations based on detected diseases. |  |  | | --- | |  | |
| PDS-06 | System | The system **shall** store historical records of disease diagnoses for future reference. |
| PDS-07 | Farmer | |  | | --- | | The system **shall** allow farmers to create and manage their user profiles. |  |  | | --- | |  | |
| PDS-08 | Agricultural Expert | |  | | --- | | The system **shall** enable agricultural experts to provide feedback on disease detection results. |  |  | | --- | |  | |
| PDS-09 | System | |  | | --- | | The system **shall** ensure the security of uploaded images and user data. |  |  | | --- | |  | |
| PDS-10 | System | |  | | --- | | The system **shall** generate reports on system accuracy and performance. |  |  | | --- | |  | |
| PDS-11 | Web Interface | |  | | --- | | The system **shall** support a web-based user interface for easy access. |  |  | | --- | |  | |
| PDS-12 | System | |  | | --- | | The system **shall** allow integration with external APIs for additional agricultural data. |  |  | | --- | |  | |
| PDS-13 | System | |  | | --- | | The system **shall** include error-handling mechanisms for failed image uploads. |  |  | | --- | |  | |
| PDS-14 | System | |  | | --- | | The system **shall** support real-time processing of images for instant disease detection. |  |  | | --- | |  | |
| PDS-15 | System | |  | | --- | | The system **shall** provide a confidence score for each disease prediction. |  |  | | --- | |  | |
| PDS-16 | Farmer | |  | | --- | | The system **shall** allow users to provide feedback on disease diagnosis accuracy. |  |  | | --- | |  | |
| PDS-17 | Government / NGOs | The system **shall** allow government agencies and NGOs to access anonymized system data for agricultural research. |

### 1.5 Allocate Requirements

|  |  |  |
| --- | --- | --- |
| **Para #** | **Initial Requirement** | **Use Case** |
| 1.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must allow users to upload images of plant leaves. |  |  | | --- | |  | |  |  | | --- | |  | | |  | | --- | | Image Upload |  |  | | --- | |  | |
| 1.0 | |  | | --- | | The system must process the uploaded images to detect plant diseases using machine learning. |  |  | | --- | |  | | |  | | --- | | Image Processing |  |  | | --- | |  | |
| 1.0 | |  | | --- | | The system must classify plant diseases based on pre-trained models. |  |  | | --- | |  | | |  | | --- | | Disease Classification |  |  | | --- | |  | |
| 1.0 | |  | | --- | | The system must provide users with disease diagnosis results. |  |  | | --- | |  | | |  | | --- | | Disease Diagnosis |  |  | | --- | |  | |
| 1.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must suggest treatment recommendations based on detected diseases. |  |  | | --- | |  | |  |  | | --- | |  | | |  | | --- | | Treatment Recommendation |  |  | | --- | |  | |
| 1.0 | |  | | --- | | The system must store historical records of disease diagnoses for future reference. |  |  | | --- | |  | | |  | | --- | | Data Storage |  |  | | --- | |  | |
| 1.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must allow farmers to create and manage their user profiles. |  |  | | --- | |  | |  |  | | --- | |  | | |  | | --- | | User Management |  |  | | --- | |  | |
| 1.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must ensure the security of uploaded images and user data. |  |  | | --- | |  | |  |  | | --- | |  | | |  | | --- | | Data Security |  |  | | --- | |  | |
| 1.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must support a web-based user interface for easy access. |  |  | | --- | |  | |  |  | | --- | |  | | |  | | --- | | Web Interface |  |  | | --- | |  | |
| 1.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must allow integration with external APIs for additional agricultural data. |  |  | | --- | |  | |  |  | | --- | |  | | |  | | --- | | API Integration |  |  | | --- | |  | |
| 2.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must include error-handling mechanisms for failed image uploads. |  |  | | --- | |  | |  |  | | --- | |  | | |  | | --- | | Error Handling |  |  | | --- | |  | |
| 2.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must support real-time processing of images for instant disease detection. |  |  | | --- | |  | |  |  | | --- | |  | | |  | | --- | | Real-Time Processing |  |  | | --- | |  | |
| 2.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must provide a confidence score for each disease prediction. |  |  | | --- | |  | |  |  | | --- | |  | | |  | | --- | | Prediction Confidence |  |  | | --- | |  | |
| 2.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must allow users to provide feedback on disease diagnosis accuracy. |  |  | | --- | |  | |  |  | | --- | |  | | |  | | --- | | User Feedback |  |  | | --- | |  | |
| 3.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must allow government agencies and NGOs to access anonymized system data for agricultural research. |  |  | | --- | |  | |  |  | | --- | |  | | |  | | --- | | Research Access |  |  | | --- | |  | |
| 3.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must work effectively under different lighting and image quality conditions. |  |  | | --- | |  | |  |  | | --- | |  | | |  | | --- | | Robust Image Processing |  |  | | --- | |  | |
| 3.0 | |  | | --- | | The system must provide mobile access for farmers in rural areas. |  |  | | --- | |  | | Mobile Compatibility |

### 1.6 Prioritize Requirements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Para #** | **Rank** | **Initial Requirement** | **Use Case\_ID** | **Use Case** |
| 1.0 | |  | | --- | | Highest |  |  | | --- | |  | | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must allow users to upload images of plant leaves. |  |  | | --- | |  | |  |  | | --- | |  | | UC\_1 | |  | | --- | | Image Upload |  |  | | --- | |  | |
| 1.0 | Highest | |  | | --- | | The system must process the uploaded images to detect plant diseases using machine learning. |  |  | | --- | |  | | UC\_2 | |  | | --- | | Image Processing |  |  | | --- | |  | |
| 1.0 | Highest | |  | | --- | | The system must classify plant diseases based on pre-trained models. |  |  | | --- | |  | | UC\_3 | |  | | --- | | Disease Classification |  |  | | --- | |  | |
| 1.0 | Highest | |  | | --- | | The system must provide users with disease diagnosis results. |  |  | | --- | |  | | UC\_4 | |  | | --- | | Disease Diagnosis |  |  | | --- | |  | |
| 1.0 | Highest | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must suggest treatment recommendations based on detected diseases. |  |  | | --- | |  | |  |  | | --- | |  | | UC\_5 | |  | | --- | | Treatment Recommendation |  |  | | --- | |  | |
| 1.0 | High | |  | | --- | | The system must store historical records of disease diagnoses for future reference. |  |  | | --- | |  | | UC\_6 | |  | | --- | | Data Storage |  |  | | --- | |  | |
| 1.0 | High | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must allow farmers to create and manage their user profiles. |  |  | | --- | |  | |  |  | | --- | |  | | UC\_7 | |  | | --- | | User Management |  |  | | --- | |  | |
| 1.0 | High | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must ensure the security of uploaded images and user data. |  |  | | --- | |  | |  |  | | --- | |  | | UC\_8 | |  | | --- | | Data Security |  |  | | --- | |  | |
| 1.0 | High | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must support a web-based user interface for easy access. |  |  | | --- | |  | |  |  | | --- | |  | | UC\_9 | |  | | --- | | Web Interface |  |  | | --- | |  | |
| 1.0 | Medium | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must allow integration with external APIs for additional agricultural data. |  |  | | --- | |  | |  |  | | --- | |  | | UC\_10 | |  | | --- | | API Integration |  |  | | --- | |  | |
| 2.0 | Medium | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must include error-handling mechanisms for failed image uploads. |  |  | | --- | |  | |  |  | | --- | |  | | UC\_11 | |  | | --- | | Error Handling |  |  | | --- | |  | |
| 2.0 | Medium | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must support real-time processing of images for instant disease detection. |  |  | | --- | |  | |  |  | | --- | |  | | UC\_12 | |  | | --- | | Real-Time Processing |  |  | | --- | |  | |
| 2.0 | Medium | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must provide a confidence score for each disease prediction. |  |  | | --- | |  | |  |  | | --- | |  | | UC\_13 | |  | | --- | | Prediction Confidence |  |  | | --- | |  | |
| 2.0 | Medium | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must allow users to provide feedback on disease diagnosis accuracy. |  |  | | --- | |  | |  |  | | --- | |  | | UC\_14 | |  | | --- | | User Feedback |  |  | | --- | |  | |
| 3.0 | Low | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must allow government agencies and NGOs to access anonymized system data for agricultural research. |  |  | | --- | |  | |  |  | | --- | |  | | UC\_15 | |  | | --- | | Research Access |  |  | | --- | |  | |
| 3.0 | Low | |  |  |  | | --- | --- | --- | | |  | | --- | | The system must work effectively under different lighting and image quality conditions. |  |  | | --- | |  | |  |  | | --- | |  | | UC\_16 | |  | | --- | | Robust Image Processing |  |  | | --- | |  | |
| 3.0 | Low | |  | | --- | | The system must provide mobile access for farmers in rural areas. |  |  | | --- | |  | | UC\_17 | Mobile Compatibility |

### 1.7 Requirements Trace-ability Matrix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr.#** | **Para #** | **System Specification Text** | **Build** | **Use Case** | **Category** |
| 1 | 1.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system “shall” must allow users to upload images of plant leaves. |  |  | | --- | |  | |  |  | | --- | |  | | B1 | |  | | --- | | Image Upload |  |  | | --- | |  | | |  | | --- | | **Core Business Function** | |
| 2 | 1.0 | |  | | --- | | The system “shall” must process the uploaded images to detect plant diseases using machine learning. |  |  | | --- | |  | | B1 | |  | | --- | | Image Processing |  |  | | --- | |  | | |  | | --- | |  | | **Core Business Function** | |  |  | | --- | |  | |
| 3 | 1.0 | |  | | --- | | The system “shall” must classify plant diseases based on pre-trained models. |  |  | | --- | |  | | B1 | |  | | --- | | Disease Classification |  |  | | --- | |  | | |  | | --- | | **Core Business Function** | |
| 4 | 1.0 | |  | | --- | | The system “shall” must provide users with disease diagnosis results. |  |  | | --- | |  | | B1 | |  | | --- | | Disease Diagnosis |  |  | | --- | |  | | |  | | --- | |  | | **Core Business Function** | |  |  | | --- | |  | |
| 5 | 1.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system “shall” must suggest treatment recommendations based on detected diseases. |  |  | | --- | |  | |  |  | | --- | |  | | B1 | |  | | --- | | Treatment Recommendation |  |  | | --- | |  | | Value-Added Feature |
| 6 | 1.0 | |  | | --- | | The system “shall” must store historical records of disease diagnoses for future reference. |  |  | | --- | |  | | B1 | |  | | --- | | Data Storage |  |  | | --- | |  | | |  | | --- | | **Business Data Management** |  |  | | --- | |  | |
| 7 | 1.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system “shall” must allow farmers to create and manage their user profiles. |  |  | | --- | |  | |  |  | | --- | |  | | B1 | |  | | --- | | User Management |  |  | | --- | |  | | |  | | --- | | **User Engagement** |  |  | | --- | |  | |
| 8 | 1.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system “shall” must ensure the security of uploaded images and user data. |  |  | | --- | |  | |  |  | | --- | |  | | B1 | |  | | --- | | Data Security |  |  | | --- | |  | | |  | | --- | | **Regulatory & Compliance** |  |  | | --- | |  | |
| 9 | 1.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system “shall” must support a web-based user interface for easy access. |  |  | | --- | |  | |  |  | | --- | |  | | B1 | |  | | --- | | Web Interface |  |  | | --- | |  | | |  | | --- | | **Accessibility & Usability** |  |  | | --- | |  | |
| 10 | 1.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system “shall” must allow integration with external APIs for additional agricultural data. |  |  | | --- | |  | |  |  | | --- | |  | | B1 | |  | | --- | | API Integration |  |  | | --- | |  | | |  | | --- | | **Business Expansion & Collaboration** |  |  | | --- | |  | |
| 11 | 2.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system “shall” must include error-handling mechanisms for failed image uploads. |  |  | | --- | |  | |  |  | | --- | |  | | B1 | |  | | --- | | Error Handling |  |  | | --- | |  | | |  | | --- | | **System Reliability** |  |  | | --- | |  | |
| 12 | 2.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system “shall” must support real-time processing of images for instant disease detection. |  |  | | --- | |  | |  |  | | --- | |  | | B1 | |  | | --- | | Real-Time Processing |  |  | | --- | |  | | |  | | --- | | **Performance Enhancement** |  |  | | --- | |  | |
| 13 | 2.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system “shall” must provide a confidence score for each disease prediction. |  |  | | --- | |  | |  |  | | --- | |  | | B1 | |  | | --- | | Prediction Confidence |  |  | | --- | |  | | |  | | --- | | **Trust & Accuracy** |  |  | | --- | |  | |
| 14 | 2.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system “shall” must allow users to provide feedback on disease diagnosis accuracy. |  |  | | --- | |  | |  |  | | --- | |  | | B1 | |  | | --- | | User Feedback |  |  | | --- | |  | | |  | | --- | | **Public Benefit & Policy Making** |  |  | | --- | |  | |
| 15 | 3.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system “shall” must allow government agencies and NGOs to access anonymized system data for agricultural research. |  |  | | --- | |  | |  |  | | --- | |  | | B1 | |  | | --- | | Research Access |  |  | | --- | |  | | |  | | --- | | **Public Benefit & Policy Making** |  |  | | --- | |  | |
| 16 | 3.0 | |  |  |  | | --- | --- | --- | | |  | | --- | | The system “shall” must work effectively under different lighting and image quality conditions. |  |  | | --- | |  | |  |  | | --- | |  | | B1 | |  | | --- | | Robust Image Processing |  |  | | --- | |  | | **Market Expansion** |
| 17 | 3.0 | |  | | --- | | The system “shall” must provide mobile access for farmers in rural areas. |  |  | | --- | |  | | B1 | Mobile Compatibility | **Market Expansion** |

### 1.9 High Level Usecase Diagram

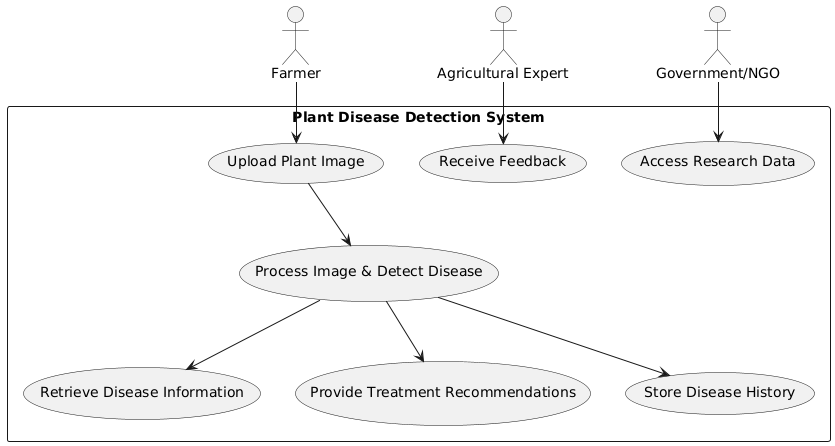
A use case scenario is a visual description, typically written in structured English or point form, of a potential business situation that a system may or may not be able to handle.

A use case defines a goal-oriented set of interactions between external actors and the system under consideration.

A use case is initiated by a user with a particular goal in mind, and completes successfully when that goal is satisfied. It describes the sequence of interactions between actors and the system necessary to deliver the service that satisfies the goal. It also includes possible variants of this sequence, e.g., alternative sequences that may also satisfy the goal, as well as sequences that may lead to failure to complete the service because of exceptional behavior, error handling, etc. The system is treated as a “black box”, and the interactions with system, including system responses, are as perceived from outside the system.

Thus, use cases capture who (actor) does what (interaction) with the system, for what purpose (goal), without dealing with system internals. A complete set of use cases specifies all the different ways to use the system, and therefore defines all behavior required of the system, bounding the scope of the system.

Generally, use case steps are written in an easy-to-understand structured narrative using the vocabulary of the domain. This is engaging for users who can easily follow and validate the use cases, and the accessibility encourages users to be actively involved in defining the requirements.

****