Thadomal Shahani Engineering College

Bandra (W.), Mumbai - 400 050.

CERTIFICATE

Certify that Mr./~~Miss~~ Aman Kanlupia

of Computers Department, Semester V with

Roll No. 2103084 has completed a course of the necessary

experiments in the subject SE under my

supervision in the **Thadomal Shahani Engineering College**

Laboratory in the year 2023 - 2024

Ansari 25/10/23

Teacher In- Charge

Head of the Department

Date 25/10/23

Principal

THADOMAL SHAHANI TSEC ENGINEERING SOLLEGE

CONTENTS

AMAN KARLUPIA
C21
2103084

EXPERIMENT NO 1:

GROUP PROJECT TITLE: Disease Prediction System

Group Members:

1)Chaitanya Kakade-2103076
2)Aman Karlupia-2103084

EXPERIMENT-1

AIM: To write a detailed Problem Statement for a case study and justifying why the chosen model would be best suited.

AMAN KARLUPIA
C21
2103084

TITLE: Disease Prediction System

PROBLEM STATEMENT:

DocBot, a disease prediction system that helps predict diseases with a high accuracy after providing symptoms as a set of input.The system has been developed by making use of machine learning models and has been deployed using a graphical user interface(GUI) by making use of Tkinter in Python.

Accurate and early disease prediction plays a pivotal role in the healthcare industry that helps in producing timely treatments.However, diagnosing a disease can be a complex procedure that requires a combination of medical knowledge and clinical expertise. Improving and tackling real life issues served as the basis for the development of the above said software.Furthermore,Artificial Intelligence and Machine Learning techniques have shown promising results in healthcare industries,thereby inculcating faith in doctors and patients.

The software has been built majorly in python programming language and the model has been trained on an available dataset which consists of 120 diseases and several symptoms corresponding to each of the disease.Then, these parameters have been used to train the model using several algorithms like Gaussian Naive Bayes,Decision Tree and Random Forest Classifier.The model has then been deployed using GUI by making use of Tkinterface. We ask the user to enter basic health details and then the user must enter atleast three symptoms he/she is suffering with.Maximum number of symptoms the user can give as input is five.Patient data is then stored on the local database.

There is a scope for further improvement by making use of a larger dataset,thereby enhancing the accuracy of the ML models. Deep Learning techniques like different types of neural networks can be incorporated to increase the machines ability to think like doctors.Moreover, the system can be used to fetch in depth information about a patient like the intensity of suffering of a particular kind of symptom,etc.Moreover the deployment can further be enhanced by deploying the model using a fully functional website so that it can be accessible worldwide and can have improved user interface.

SOFTWARE MODEL:

AMAN KARLUPIA
C21
2103084

Traditional Model:Waterfall Model.

For the development of the above mentioned project,with respect to the traditional model,the *waterfall* model will be best suited.In the Waterfall model, the development process progresses linearly through several distinct phases, and each phase must be completed before moving on to the next one.
The Waterfall model is a traditional software development methodology characterized by its sequential and linear approach to project management. In this model, the development process flows through distinct phases, each building upon the completion of the previous one. It is particularly suitable when project requirements are well-defined and stable, and there is a clear understanding of the end goals from the outset.

The model consists of several phases, including requirements gathering, system design, implementation, testing, integration, and deployment. Each phase is completed in a strict order, with no overlap between phases. The Waterfall model emphasizes thorough documentation at each step, making it well-suited for projects where traceability, compliance, and detailed documentation are paramount. However, it can be less adaptive to changes in requirements or unexpected insights during development. While the Waterfall model provides a structured framework for planning and execution, its rigid nature can sometimes limit the ability to incorporate evolving feedback and iterative adjustments often necessary in dynamic and complex projects like machine learning.

Waterfall model is best suited because of following reasons:

1. Clear Phases and Structure: The Waterfall model's distinct phases—requirements, design, implementation, testing, deployment, and maintenance—provide a clear structure for project development. This can be suitable when you have a well-defined scope and known set of requirements, which might be the case if the disease prediction project has very specific symptom-input patterns and clear disease outcomes.
2. Predictable Timeline and Budget: Waterfall's linear progression makes it easier to estimate timelines and allocate resources. This can be advantageous for projects where strict adherence to a predetermined schedule and budget is critical.
3. Documentation and Traceability: The Waterfall model emphasizes documentation at each phase, ensuring a clear record of decisions, design choices, and implementation details. This can be beneficial in regulated industries where traceability and documentation are essential, such as healthcare.
4. Minimal Scope Changes: Waterfall discourages frequent changes once a phase is completed. If the disease prediction project's symptom-input patterns and disease outcomes are well-defined and stable, this can help prevent scope creep.
5. High-Quality End Product: With thorough testing and review in each phase, the Waterfall model aims to deliver a high-quality end product. If the project's requirements are stable and unlikely to change, this approach can be effective in ensuring a robust final model.

AMAN KARLUPIA
C21
2103084

Agile Model:Scrum.

The best suited software model for the above developed project is the Agile software development technique,specifically *Scrum*.Scrum usually involves several modern software engineering practices, like continuous delivery/integration organized in sprints, plannings, estimations, and daily meetings.

Machine learning projects are different to typical software engineering projects. ML projects are likely to fail, as it's unclear if the data includes the information needed to achieve the goal, or if a method exists that can model the data. On the other side such projects can return high rewards e.g., automate costly processes, introduce innovation, or automate decisions. Compared to typical product iterations, machine learning projects have limited predictability. They are also iterative projects, but they require open project phases and need the flexibility to make major changes.

1) Flexibility and Adaptability:

Machine Learning (ML) projects in healthcare often face evolving requirements and uncertainty due to the dynamic nature of disease-related factors, research findings, and user needs. Agile methodologies, such as Scrum, are well-suited for such projects because they embrace flexibility and adaptability throughout the development process. Agile advocates for iterative development and continuous feedback loops, allowing the project team to respond quickly to changes.

In the context of healthcare, new research findings might lead to the modification of existing algorithms or the inclusion of additional data features. Evolving disease-related factors or changes in patient demographics could also impact the model's performance. By following Agile principles, the development team can adjust the project's scope, priorities, and deliverables at the end of each sprint based on the latest insights and feedback.

2) Frequent Updates and Iterations:

Agile methodologies emphasize delivering working increments at the end of each sprint, typically lasting two to four weeks. This incremental approach enables ML-based healthcare projects to quickly incorporate new findings and features. It also allows the project team to validate their work more frequently, reducing the risk of developing a product that does not meet user expectations.

Frequent updates and iterations also provide an opportunity to detect and address issues early in the development process. By continuously integrating feedback from stakeholders, including medical professionals and end-users, the ML model can be refined and improved at a rapid pace.

3) User-Centric Development:

In healthcare, the effectiveness and impact of ML models heavily depend on how well they address user needs and align with the requirements of medical professionals and patients. Agile methodologies, particularly Scrum, prioritize customer collaboration and user feedback.

Incorporating the perspectives of medical practitioners, healthcare administrators, and patients throughout the development process helps the ML project team gain valuable insights. Understanding the end-users' pain points and preferences ensures that the ML model is designed to solve real-world problems effectively.

User-centric development also encourages a more empathetic approach towards understanding the specific challenges faced by healthcare professionals in diagnosis, treatment, and patient care. This, in turn, leads to the development of ML models that are more likely to be adopted and integrated into the existing healthcare ecosystem.

4) Rapid Time-to-Market:

Agile methodologies, by their nature, promote quicker development cycles and early deployment of functional increments. This accelerated development process, combined with continuous feedback and validation, contributes to a rapid time-to-market for ML-based healthcare solutions.

In healthcare, where timely intervention and disease prediction can significantly impact patient outcomes, Agile's ability to deliver working solutions in shorter cycles is crucial. Early deployment of ML models that show promise can lead to better patient care, improved treatment plans, and even life-saving interventions.

It is important to note that while Agile brings several advantages, including flexibility and adaptability, it should be used with consideration of the specific context and requirements of each ML-based healthcare project. Adherence to relevant regulations, privacy concerns, and

ethical considerations must always be at the forefront of the development process in the healthcare domain.

AMAN KARLUPIA
C21
2103084

EXPERIMENT NO 2:

GROUP PROJECT TITLE: Application of Agile Process Model on the project(JIRA).

Group Members:

1)Chaitanya Kakade-2103076

2)Aman Karlupia-2103084

Name:Chaitanya Kakade
Batch:C21
Roll No.:2103076

AIM: Application of Agile Process Model on the project(JIRA).

THEORY:

Kanban is a popular project management and workflow optimization methodology that originated in Japan and was first implemented by Toyota in the 1940s as part of their manufacturing process. It has since been adapted and applied to various industries and processes, including software development, product design, and even personal task management.

The term "Kanban" is Japanese and can be translated to "visual signal" or "card." The fundamental idea behind Kanban is to visualize and manage work processes, making it easier to understand, control, and optimize them. Here are the key principles and

components of Kanban:

1. Visualizing Work: Kanban relies on visual representations of work items and their progress. These are typically represented as cards on a physical board or digital equivalents on a digital board or software tool. Each card represents a specific task, project, or work item.

2. Work in Progress (WIP) Limits: One of the core tenets of Kanban is limiting the number of tasks or items that can be in progress at any given time. This helps prevent overloading the team and ensures a smooth and controlled workflow. WIP limits are set for each stage of the process.

3. Flow: The goal of Kanban is to maintain a steady flow of work through the system, from initiation to completion, without bottlenecks or excessive waiting. This promotes efficiency and helps deliver value to customers more consistently.

4. Pull System: Work items are pulled through the process based on actual demand, rather than being pushed into the system. When a team member completes a task or a slot becomes available within a WIP limit, the team pulls the next highest-priority item from the backlog.

5. Continuous Improvement: Kanban encourages continuous process improvement through frequent inspection and adaptation. Teams regularly review their processes, identify bottlenecks, and make incremental changes to enhance efficiency and productivity.

6. Collaborative Approach: Kanban promotes collaboration and open communication within a team. Team members can easily see the status of work items, which fosters transparency and helps in identifying and resolving issues.

Kanban is highly flexible and can be adapted to various workflow scenarios. It doesn't prescribe specific roles or ceremonies but instead provides a framework that can be
Name:Chaitanya Kakade
Batch:C21
Roll No.:2103076
customized to suit the needs of a particular project or organization. This adaptability has contributed to its popularity in both software development and other knowledge work domains.

Agile Structure:

Agile is a project management and software development approach that emphasizes flexibility, collaboration, customer feedback, and the rapid delivery of working products or solutions. Agile methodologies are used to manage complex projects and are especially popular in the software development industry. Here's a detailed explanation of the Agile

AMAN KARLUPIA
C21
2103084

structure:

1. Iterative and Incremental Development: Agile projects are divided into small, manageable iterations or time-bound phases, often called sprints or iterations. Each iteration typically lasts from one to four weeks. During each iteration, a subset of features or user stories is developed, tested, and delivered. This approach allows for regular assessment and adaptation, reducing the risk of project failure.

2. Cross-Functional Teams: Agile teams are typically small, cross-functional groups that include various roles such as developers, testers, designers, and business analysts. These teams are responsible for all aspects of the project, from planning and design to development and testing.

3. Customer-Centric: Agile is highly customer-centric. It prioritizes delivering value to the customer through the early and continuous delivery of functional software. Customer feedback is actively sought and incorporated throughout the project, which ensures that the end product meets the customer's needs and expectations.

4. User Stories: Agile projects are often managed using user stories, which are brief, plain-language descriptions of a feature or functionality from the end user's perspective. These user stories serve as the basis for defining and prioritizing work in the backlog.

5. Backlog: The product backlog is a prioritized list of all the features, enhancements, and fixes that are to be developed during the project. It's a dynamic document that evolves as the project progresses, with new items added and priorities adjusted based on customer feedback and changing requirements.

6. Sprint Planning: At the beginning of each sprint, the team holds a sprint planning meeting to select a set of user stories or backlog items to work on during that iteration. These items are chosen based on their priority and the team's capacity to complete them within the sprint's timeframe.

7. Daily Standup (Scrum): Many Agile frameworks, such as Scrum, include daily stand-up meetings, where team members briefly share what they've worked on, what they plan to do
Name:Chaitanya Kakade
Batch:C21
Roll No.:2103076
next, and any blockers or challenges they're facing. This promotes communication, collaboration, and issue resolution.

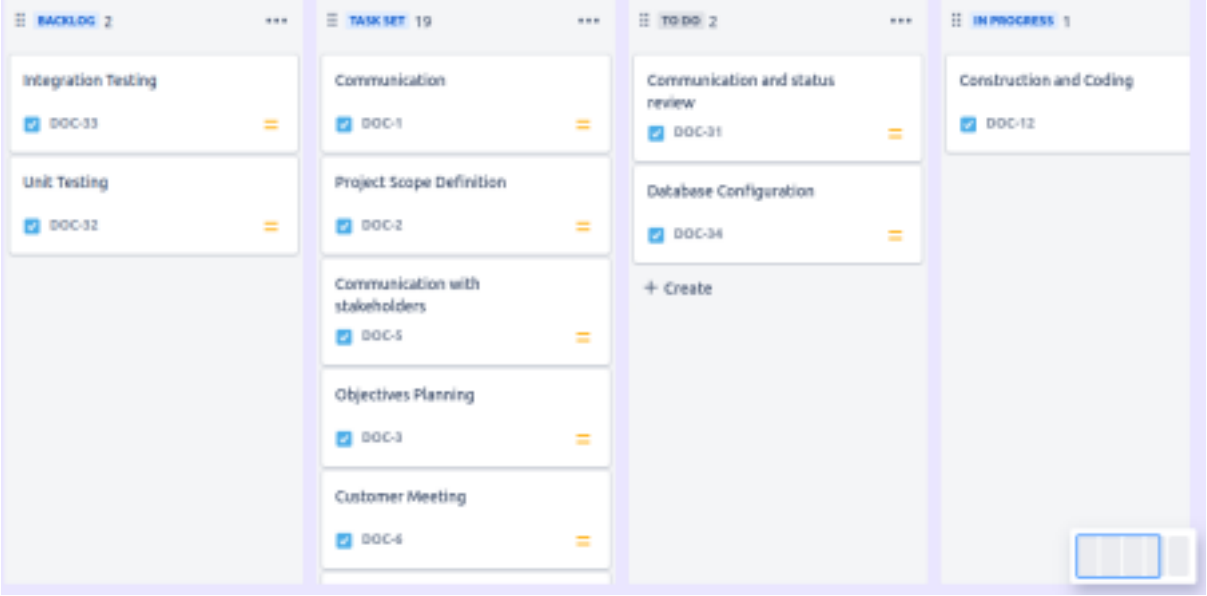
8. Sprint Review: At the end of each sprint, the team conducts a sprint review or demo where they showcase the completed work to stakeholders, including the customer. This provides an opportunity for feedback and validation, which can be used to adjust the product backlog.

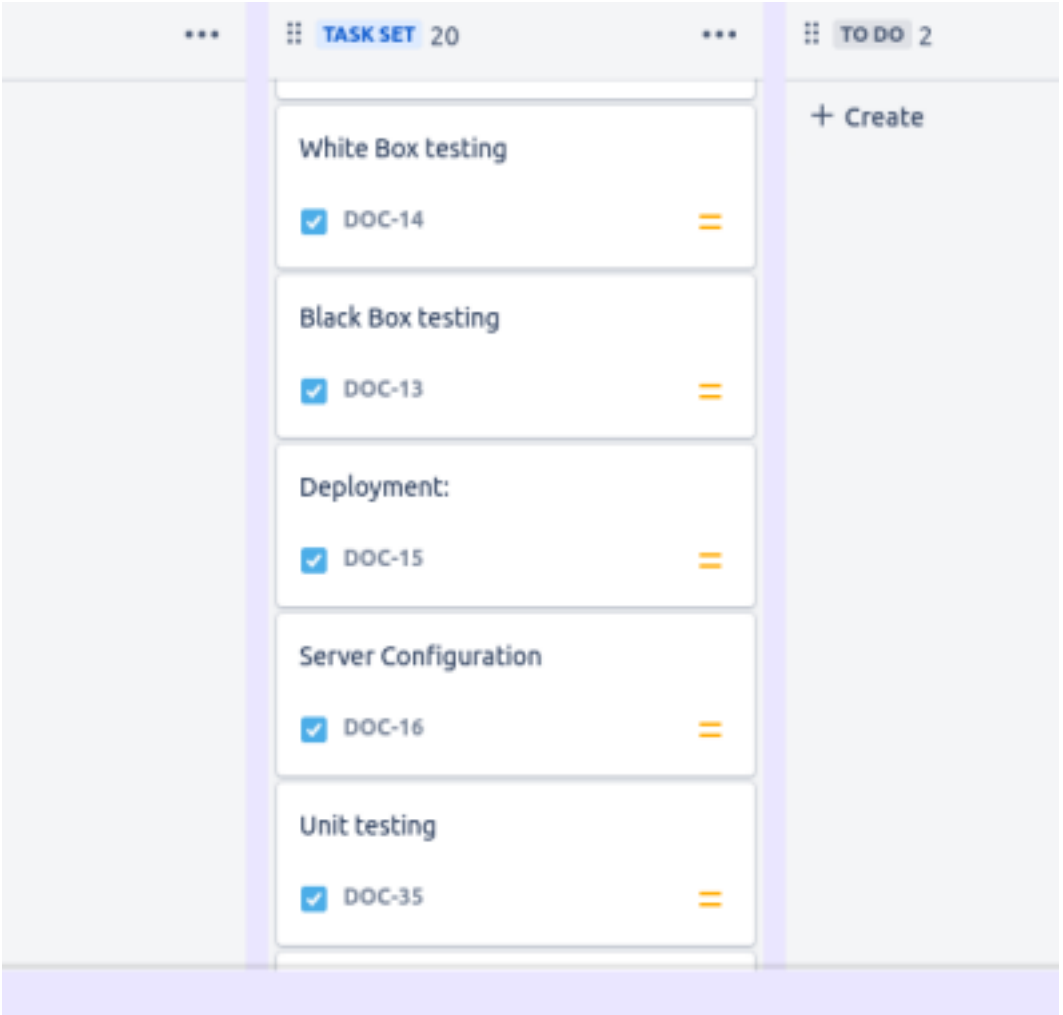
AMAN KARLUPIA
C21
2103084

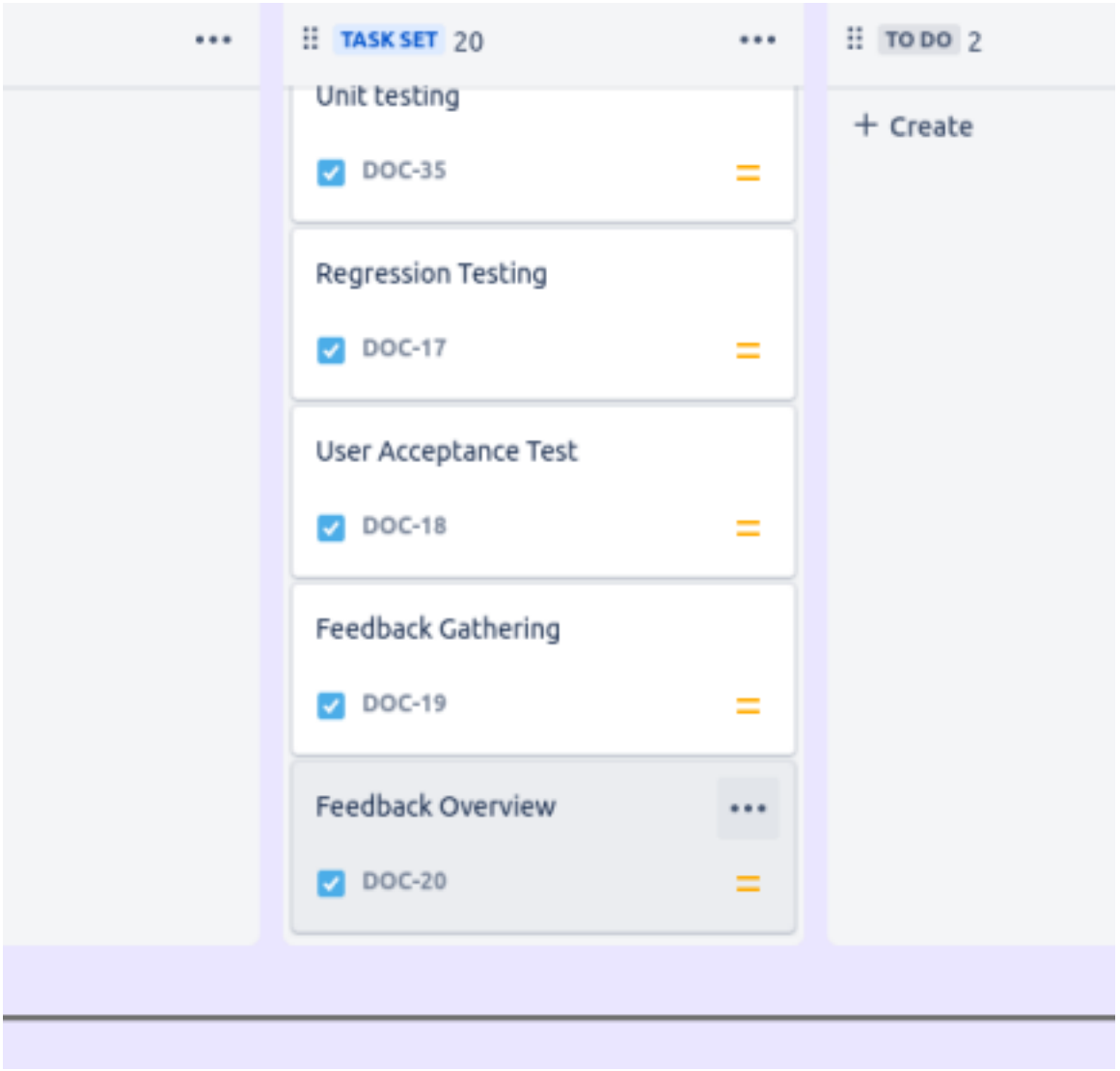
9. Retrospective: After each sprint, Agile teams hold a retrospective meeting to reflect on what went well, what didn't, and what improvements can be made. This continuous feedback loop helps the team make adjustments and improve their processes.

10. Adaptation: Agile encourages adaptability. The project and product are expected to evolve throughout the development process based on changing customer needs and market conditions. This is in contrast to traditional, plan-driven methodologies where changes are often discouraged.

11. Agile Frameworks: There are various Agile frameworks and methodologies, with Scrum, Kanban, and Extreme Programming (XP) being some of the most widely used. Each framework has its own set of practices, roles, and ceremonies, but they all share the core Agile principles of flexibility, collaboration, and customer focus.




TASK SET 20
TO DO 2
+ Create
White Box testing
DOC-14
Black Box testing
DOC-13
Deployment:
DOC-15
Server Configuration
DOC-16
Unit testing
DOC-35


TASK SET 20
TO DO 2
Unit testing
DOC-35
Regression Testing
DOC-17
User Acceptance Test
DOC-18
Feedback Gathering
DOC-19
Feedback Overview
DOC-20
Create

AMAN KARLUPIA
C21
2103084

EXPERIMENT NO 3:

GROUP PROJECT TITLE: Develop SRS document in IEEE format for the project.

Group Members:

1)Chaitanya Kakade-2103076
2)Aman Karlupia-2103084

Software Requirements Specification for Disease Prediction System

Software Requirements Specification

for

Disease Prediction System

Version 1.0 approved

Prepared by

Chaitanya Kakade-2103076

Aman Karlupia-2103084

Thadomal Shahani Engineering College

2nd August,2023

AMAN KARLUPIA
C21
2103084

System Table of Contents

Revision History

Name Date Reason For Changes Version

1. Introduction

1.1 Purpose

The primary purpose of the ***"DocBot"*** disease prediction system is to provide a reliable and efficient tool for accurately predicting diseases based on input symptoms. By harnessing the power of machine learning models and a user-friendly graphical user interface (GUI), the system aims to assist healthcare professionals and users in making informed decisions, enabling timely treatments and improving patient outcomes. Through its utilization of advanced algorithms, ***"DocBot"*** seeks to bridge the gap between medical expertise and technological innovation, enhancing disease diagnosis and contributing to the broader landscape of healthcare advancements.

1.2 Document Conventions

Important points have been underlined to provide emphasis. Headings and Subheadings have been written in bold font to provide emphasis. The points in all sections have been written in the order of their priority, from higher priority points to lower priority points, so that important points are not missed out. Abbreviations are used in some places which will be

understood by the developers of the application.

1.3 Intended Audience and Reading Suggestions

The Software Requirements Specification (SRS) document for the "DocBot" disease prediction system is designed to cater to a diverse audience, each with specific roles and expertise. Primary stakeholders include developers, testers, healthcare professionals, project managers, domain experts, and end users. Developers will engage with the document to gain a comprehensive understanding of the system's technical intricacies and implementation details. Testers will refer to it to align their validation and verification efforts with the specified requirements. Healthcare professionals will explore the system's accuracy and potential applicability to medical diagnosis. Project managers will find insights into resource allocation and project milestones. Domain experts will contribute their medical knowledge to ensure the system's clinical accuracy. End users, both patients and medical personnel, will evaluate the system's user-friendliness and diagnostic capabilities.

1.4 Product Scope

The scope of the disease prediction system encompasses the development of a machine learning-based software solution that accurately predicts diseases based on user-input symptoms. This software includes a user-friendly graphical user interface (GUI) developed with Tkinter, enabling users to enter symptoms and receive timely predictions. The system aims to bridge medical expertise and technological innovation, offering valuable assistance to healthcare professionals and users in making informed decisions and facilitating early disease diagnosis and treatment. The software's primary focus is on disease prediction and its integration within the existing healthcare landscape.
Software Requirements Specification for Disease Prediction

System 1.5 References

The development of disease prediction system's software requirements specification (SRS) draws from a range of resources including academic research such as**"Sneha Grampurohit and Chetan Sagarnal, "Disease Prediction using Machine Learning Algorithms", International Conference for Emerging Tehnology (INCET),2020."**
Valuable dataset from Kaggle (https://www.kaggle.com/datasets/rabisingh/symptom-checker?select=Training.csv) for training and validation. Python programming language and its Scikit-learn library (Python Software Foundation, 2021; Scikit-learn, 2021) were instrumental in developing and implementing machine learning algorithms. The graphical user interface (GUI) component was constructed using Tkinter (Tkinter Documentation, 2021).

2. Overall Description

2.1 Product Perspective

The "DocBot" disease prediction system functions as an independent software application within the broader healthcare ecosystem. It serves as a specialized tool that employs machine learning algorithms to predict diseases based on user-input symptoms. While it operates as a standalone system, it can potentially interface with electronic health record systems in the future, enhancing its integration with clinical workflows and offering healthcare professionals an additional diagnostic aid.

2.2 Product Functions

The disease prediction system encompasses the following primary functions, which are elaborated upon in Section 3 of the SRS document:

- Symptom Input: Users can input sets of symptoms into the system. ● Disease Prediction: The system utilizes machine learning models (RFC, GNB, DT) to predict diseases based on the provided symptoms.
- User-Friendly Interface: The graphical user interface (GUI) guides users through symptom input and displays the predicted diseases.
- Data Storage: User-entered symptom data is stored in a local database for future reference.
- Accuracy Assessment: Users can assess the accuracy of predictions by comparing them with actual outcomes.
- Potential Enhancements: The system can be extended to include deeper patient information, incorporate larger datasets, and deploy the model via a fully functional website.

System 2.3 User Classes and Characteristics

The "DocBot" disease prediction system addresses a range of user classes, encompassing Patients, Medical Professionals, Developers, and Testers. Patients, with varying degrees of technical familiarity, utilize the system for accurate disease predictions based on symptom input. Medical Professionals, armed with extensive medical knowledge, validate predictions. Developers, proficient in technical aspects, ensure the system's development and refinement. Testers, applying their testing expertise, verify accuracy and performance. Prioritizing Patients' needs, the system seamlessly caters to the unique characteristics and requirements of each user class, contributing to its overall usability and reliability.

2.4 Operating Environment

The disease prediction system operates within a specific environment, encompassing hardware, operating systems, and software components. This environment is carefully chosen to ensure optimal functionality and seamless performance of the system. The operating environment is as follows:

1.Hardware Platform: The system is designed to function on standard personal computers and laptops, encompassing a range of processor types and memory capacities. 2.Operating System: The system is compatible with major operating systems, including Windows (Windows 10 and later versions), macOS (macOS 10.12 and later

versions), and Linux distributions (Ubuntu 18.04 and later versions).
3.Software Dependencies: The system relies on several software components to function effectively:
- Python (version 3.6 and later): The primary programming language used for system development and implementation.
- Scikit-learn library: Essential for implementing machine learning algorithms like Random Forest Classifier, Gaussian Naive Bayes, and Decision Tree.
- Tkinter library: Utilized for creating the graphical user interface (GUI) to facilitate user interactions.
- Local Database System: A local database is required for storing user-entered symptom data.

By accommodating a wide range of hardware platforms, operating systems, and software dependencies, the "DocBot" system ensures accessibility and usability across various user environments. Its adaptability to both local and web-based deployments underscores its versatility and potential for wider usage.

2.5 Design and Implementation Constraints

CO-1:
The time allotted for this project is at most 3 months.
CO-2:
The front end of the application will be made using GUI
CO-3:
MySQL will be used for the database of the application.
CO-4:The system will be in English language. Users who do not know English will face difficulties in using the system.

Software Requirements Specification for Disease Prediction

System 2.6 User Documentation

The user documentation for the disease prediction system comprises a comprehensive user manual offering detailed instructions on symptom input, disease prediction, GUI navigation, and accuracy assessment; an integrated online help resource providing real-time guidance;
interactive tutorials for self-guided learning; a FAQ section addressing common queries; and detailed release notes accompanying software updates. This documentation adheres to industry standards and is delivered in electronic formats like PDFs and web-based resources, aiming to empower users with clear, accessible, and valuable support materials for optimal system utilisation.
Proper error messages will be displayed in case the user inadvertently fills wrong information or makes any mistake while using the application.

2.7 Assumptions and Dependencies

The disease prediction system's requirements are influenced by several assumptions and dependencies. Assumptions include the availability of a reliable internet connection for potential web deployment, consistent compatibility with Python, Scikit-learn, and Tkinter

libraries, and the presence of local database systems. Dependencies involve the successful integration of machine learning models (RFC, GNB, DT) for accurate predictions, adherence to specified operating systems (Windows 10, macOS 10.12, Ubuntu 18.04), and the seamless interaction of user data with the local database. The project's effectiveness hinges on the validity of these assumptions and the unimpeded fulfilment of dependencies.

3. External Interface Requirements

3.1 User Interfaces

The "DocBot" disease prediction system's user interface (UI) is designed for optimal user interaction and engagement. The UI commences by prompting users to input essential patient information such as name, age, height, and blood group. Following this, users are guided to input a minimum of three and a maximum of five symptoms, tailoring the experience to their specific health concerns. The UI prominently displays a "Predict" button that, upon activation, triggers the system to process the provided patient data and symptoms, ultimately generating a probable disease prediction. The UI's intuitive design ensures that entering patient data and symptoms is seamless and efficient, facilitating accurate predictions for enhanced healthcare decision-making.

3.2 Hardware Interfaces

N/A

3.3 Software Interfaces

The disease prediction system interfaces with specific software components to ensure efficient functionality and seamless data exchange. It interacts with the following components:
1. Python (Version 3.6 and later): The primary programming language used for system development, hosting machine learning models, and GUI development. 2. Scikit-learn Library: Facilitates the implementation of machine learning algorithms like Random Forest Classifier, Gaussian Naive Bayes, and Decision Tree. 3. Tkinter Library: Enables the creation of the graphical user interface (GUI) for user interactions.
The data flow includes:
Symptom and Patient Data Inflow: Users provide patient information, symptoms, and inputs through the GUI.
Prediction Outflow: The system generates probable disease predictions based on user-provided data.
The system communicates with the software components using Python APIs, and data exchange adheres to standard protocols such as function calls, data serialization, and event-driven mechanisms. Shared data includes symptom and patient information, which is used for disease prediction. The software interfaces are implemented to ensure seamless interactions and accurate predictions, enhancing the overall system performance and user experience.

3.4 Communications Interfaces

N/A

4. System Features

4.1 User Information Input

4.1.1 Description and Priority

The "User Information Input" feature allows users to provide their name, age, height, and weight, which are essential for health assessment and BMI calculation. This feature is of high priority as it is the initial step in personalising the user's experience.

4.1.2 Stimulus/Response Sequences

Stimulus 1: The user selects the "Input Personal Information" option.
Response 1: The system prompts the user to enter their name, age, height, and weight.
Stimulus 2: The user enters personal information.
Response 2: The system validates and stores the personal information.

System 4.1.3 Functional Requirements

REQ-1: The system shall provide an interface for users to input their name, age, height, and weight.

REQ-2: The system shall validate the user's age to ensure it is within an acceptable range (e.g., 1 to 120 years). If the age is invalid, an error message shall be displayed.

REQ-3: The system shall validate the user's height to ensure it is within an acceptable range (e.g., 30 cm to 300 cm). If the height is invalid, an error message shall be displayed.

REQ-4: The system shall validate the user's weight to ensure it is within an acceptable range (e.g., 1 kg to 500 kg). If the weight is invalid, an error message shall be displayed.

REQ-5: The system shall provide feedback to the user after successful input of personal information.

4.2 Symptom Input and Disease Prediction

4.2.1 Description and Priority

The "Symptom Input and Disease Prediction" feature allows users to input symptoms and predicts diseases based on these symptoms using machine learning algorithms. This feature is of high priority as it constitutes the core functionality of "DocBot."

4.2.2 Stimulus/Response Sequences

Stimulus 1: The user selects the "Input Symptoms" option.

Response 1: The system prompts the user to enter symptoms.

Stimulus 2: The user enters symptoms.

Response 2: The system validates and stores the symptoms.

Stimulus 3: The system invokes machine learning algorithms (e.g., Gaussian Naive Bayes, Random Forest Classifier, Decision Tree) to predict diseases.

Response 3: The system generates a prediction based on the

algorithms. Stimulus 4: The user views the predicted disease.

Response 4: The system displays the predicted disease along with a confidence score.

4.2.3 Functional Requirements

REQ-1: The system shall accept user-provided symptom data as input for disease prediction.

REQ-2: The system shall process the symptom data using machine learning algorithms to predict diseases.

REQ-3: The system shall provide a prediction result, including the predicted disease and a confidence score.

REQ-4: If the system encounters issues during the prediction process, it shall provide an appropriate error message to the user.

REQ-5: The system shall store the prediction results, including the predicted disease, in the database for future reference.

REQ-6: The system shall allow users to request disease predictions multiple times.

REQ-7: The system shall provide clear and user-friendly explanations for the predicted diseases.

REQ-8: The system shall log prediction requests and results for auditing purposes.

4.3 Calculate BMI

4.3.1 Description and Priority

The "Calculate BMI" feature calculates the Body Mass Index (BMI) based on the user's height and weight. This feature is of medium priority, providing valuable health information to the user.

4.3.2 Stimulus/Response Sequences

Stimulus 1: The user selects the "Calculate BMI" option.

Response 1: The system prompts the user to enter their height and

weight. Stimulus 2: The user enters height and weight.

Response 2: The system calculates the BMI and displays the result.

4.3.3 Functional Requirements

REQ-1: The system shall provide an interface for users to input their height and weight for BMI calculation.

REQ-2: The system shall calculate the BMI using the formula: BMI = (Weight in kilograms) / (Height in metres)^2.

REQ-3: The system shall display the calculated BMI to the user.

REQ-4: The system shall provide an interpretation of the BMI result, indicating whether it is underweight, normal, overweight, or obese.

REQ-5: The system shall log BMI calculation requests and results for auditing purposes.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

The model should provide predictions within an acceptable response time, especially in real-time or near-real-time clinical settings. Response times should be defined and met.

5.2 Safety Requirements

Safety in a disease prediction model involves patient risk assessment, data privacy, regulatory compliance, bias mitigation, documentation, and emergency protocols to protect patients and ensure ethical, secure, and reliable healthcare outcomes.

5.3 Security Requirements

Robust security measures must be in place to protect patient data and the model itself from unauthorised access, breaches, and cyberattacks. Compliance with data protection regulations (e.g., HIPAA) is essential.

5.4 Software Quality Attributes

5.4.1. Accuracy

Ensuring precise disease predictions to aid clinical decision-making.

5.4.2. Reliability

Consistently delivering accurate predictions in various healthcare scenarios.

5.4.3. Usability

Providing a user-friendly interface for healthcare professionals to interact effectively.

5.4.4. Scalability

Handling growing data and user loads to maintain performance.

System 5.5 Business Rules

5.5.1. Patient Data Privacy

Ensure that patient data is anonymized and comply with data protection regulations (e.g., HIPAA or GDPR).

5.5.2. Clinical Validation

Validate the model's predictions against clinical standards and guidelines before implementing it in patient care.

5.5.3. Alert Thresholds

Define specific thresholds for prediction probabilities that trigger alerts to healthcare professionals.

5.5.4. Data Retention

Define policies for data retention and deletion in compliance with relevant regulations.

5.5.5. Feedback Loop

Create a mechanism for healthcare professionals to provide feedback on the model's predictions and performance.

6. Other Requirements

6.1. Resource Management

Efficiently manage computational resources such as server capacity and data storage.

6.2. Backup and Recovery

Develop robust backup and recovery procedures to protect against data loss or system failures.

6.3. Regulatory Compliance

Ensure adherence to regulatory requirements specific to medical devices, software, and healthcare data.

6.4. Localization and Language Support

Provide support for multiple languages and adapt to regional medical practices.

Appendix A: Glossary

1. Disease Prediction Model: Software for forecasting disease likelihood.
2. Clinical Validation: Ensuring model accuracy against medical standards.
3. HIPAA: U.S. healthcare data privacy law.
4. Interoperability: Integration with healthcare systems.
5. Bias Mitigation: Addressing prediction biases.

6. Redundancy: Backup for system reliability.

7. Data Drift: Changing data patterns.

8. BMI-Body Mass Index

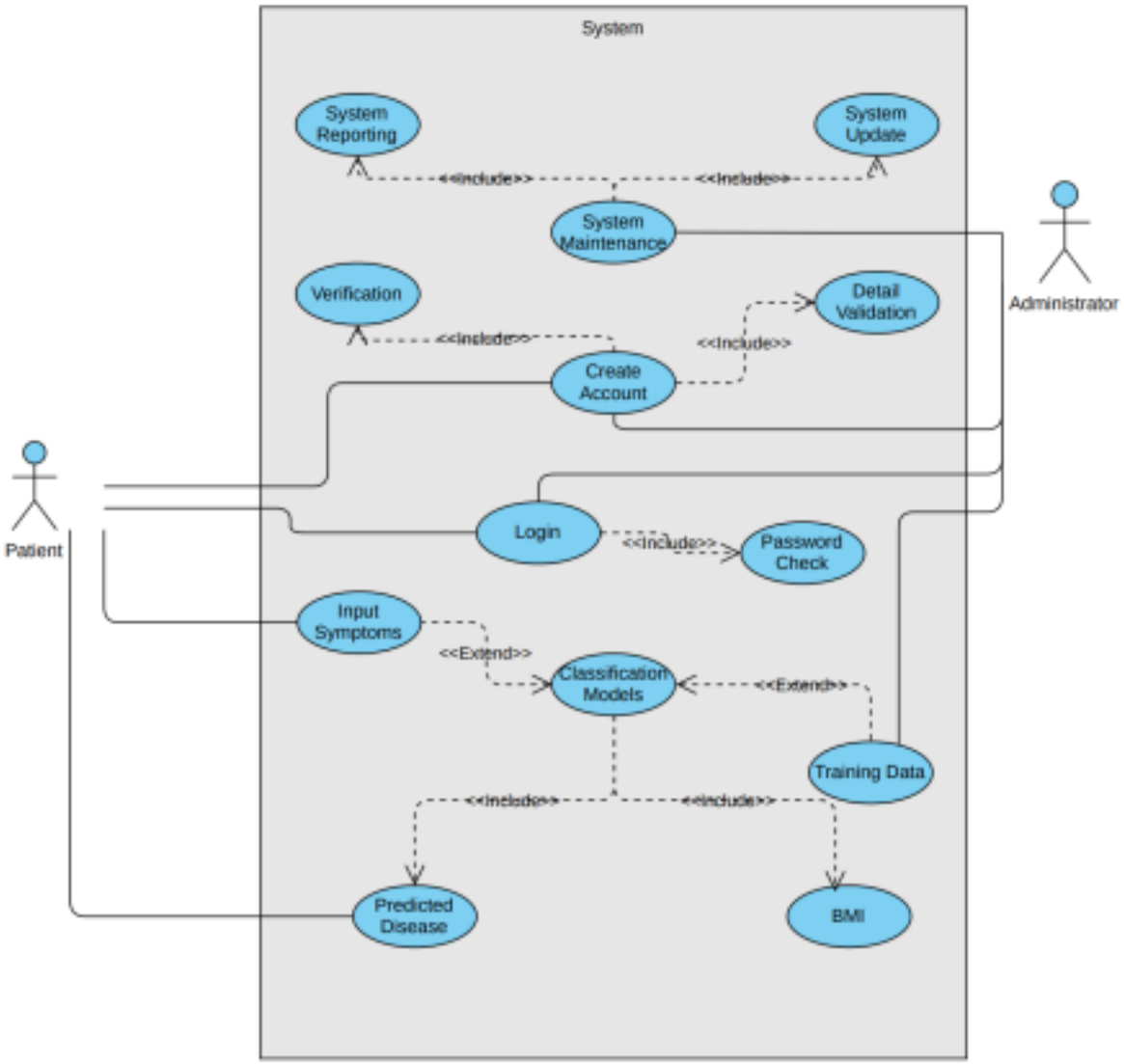
9. Cost Management: Expense optimization.

10. Crisis Response Planning: Handling emergencies.

11. Long-Term Support: Ongoing maintenance commitment.

Appendix B: Analysis Models

Use Case Diagram for the system


System
System Reporting
System Update
<<Include>>
<<Include>>
System Maintenance
Verification
Detail Validation
<<Include>>
<<Include>>
Create Account
Administrator
Patient
Login
<<Include>>
Password Check
Input Symptoms
<<Extend>>
Classification Models
<<Extend>>
Training Data
<<Include>>
<<Include>>
Predicted Disease
BMI

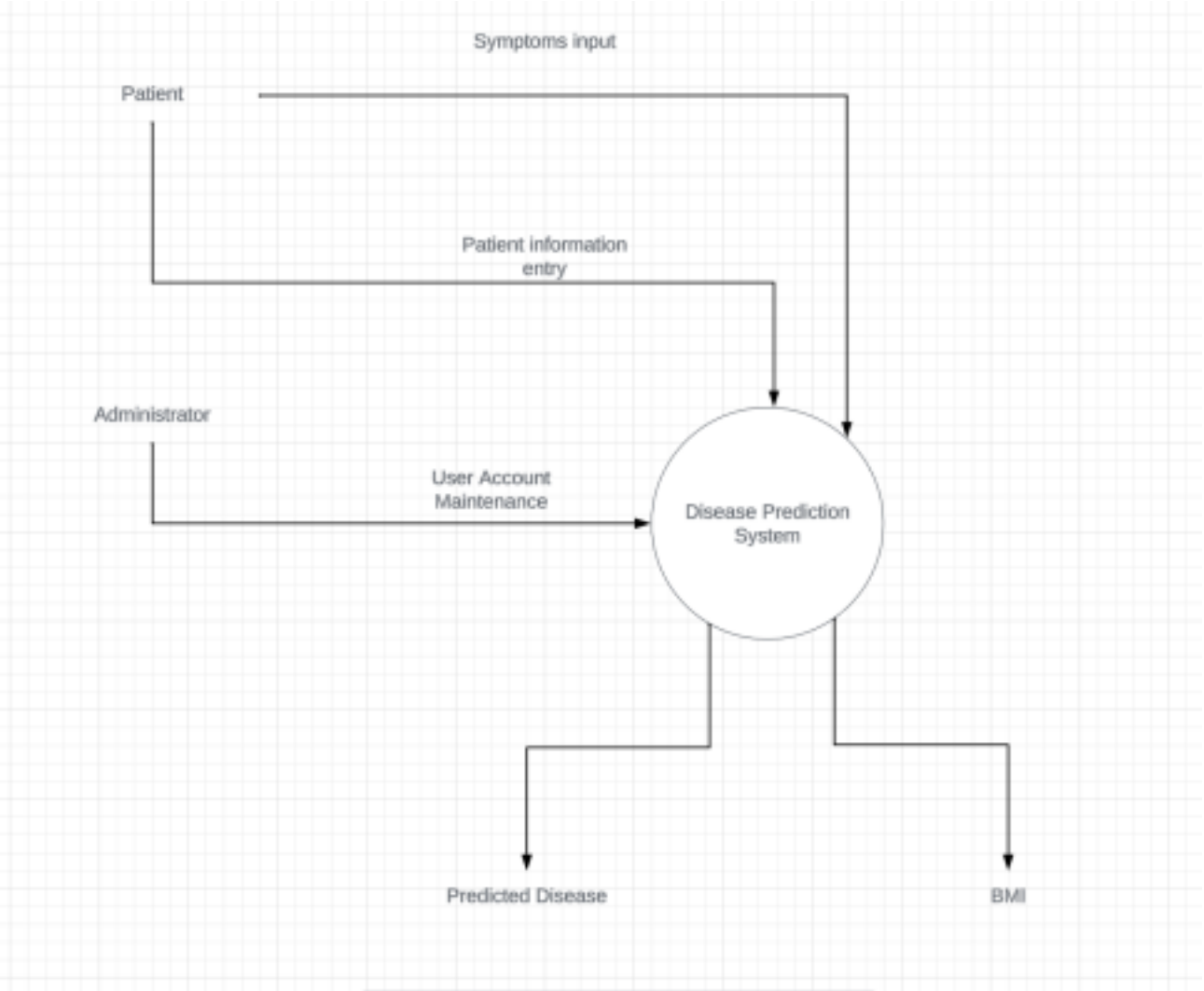
EXPERIMENT NO 4:

GROUP PROJECT TITLE: Develop Data Flow Diagram(DFD) for the project

Group Members:

1)Chaitanya Kakade-2103076
2)Aman Karlupia-2103084

Name:Chaitanya Kakade
Batch:C21
Roll_No:2103076



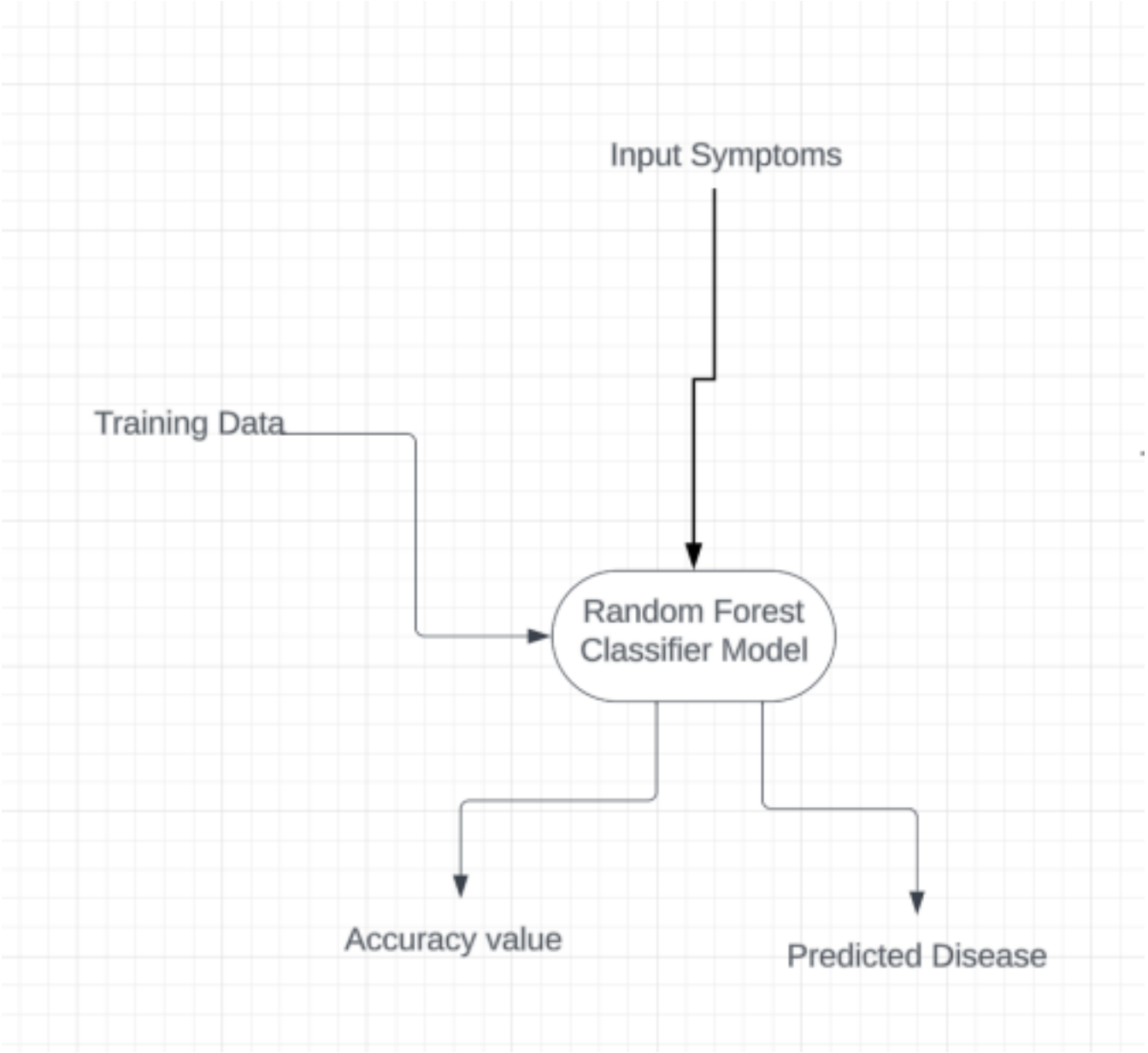
LEVEL 0 DFD

Name:Chaitanya Kakade Batch:C21 Roll_No:2103076

Name:Chaitanya Kakade Batch:C21 Roll_No:2103076

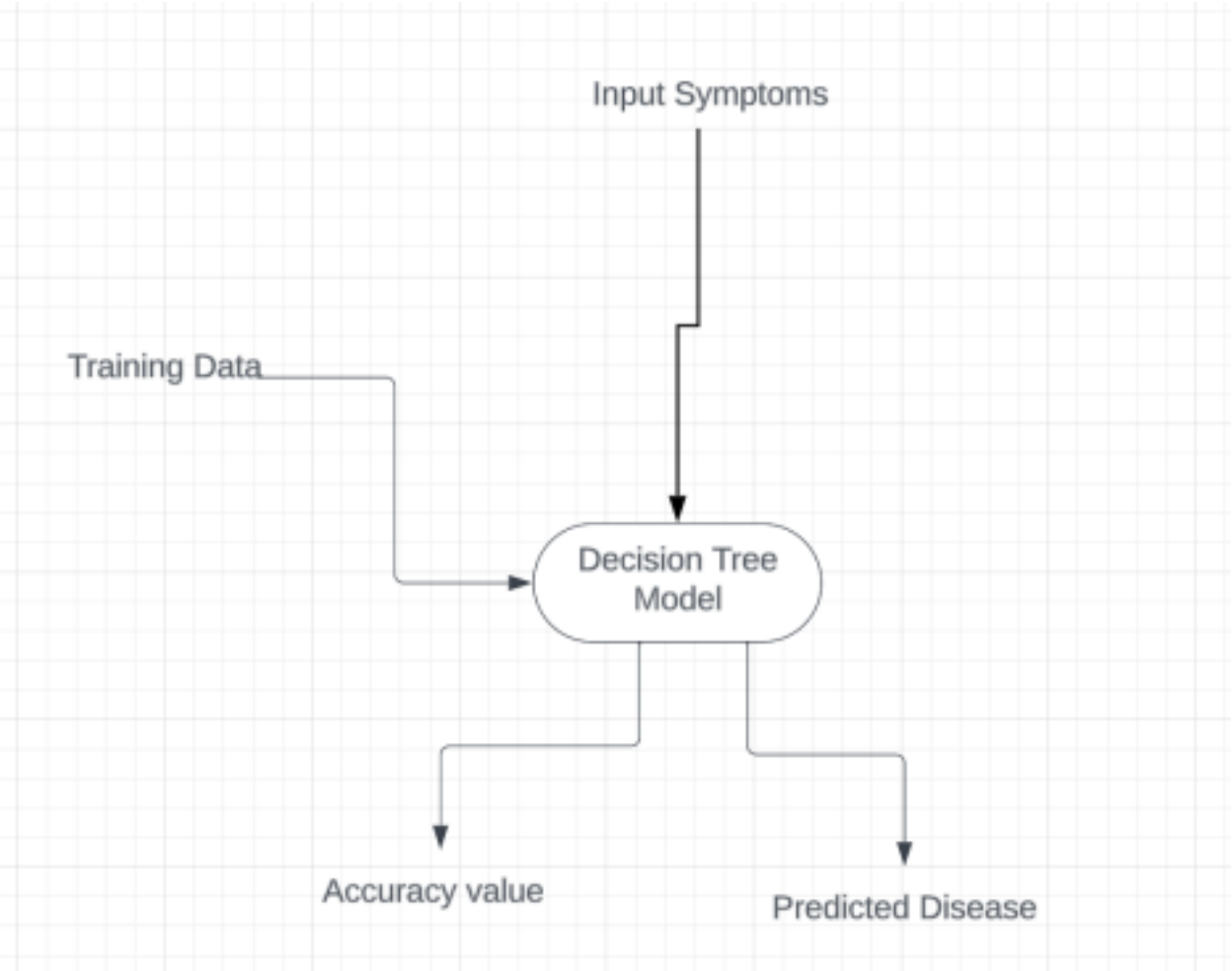
Level 2 DFDs

Symptoms’ Prediction and Management(Model 1)



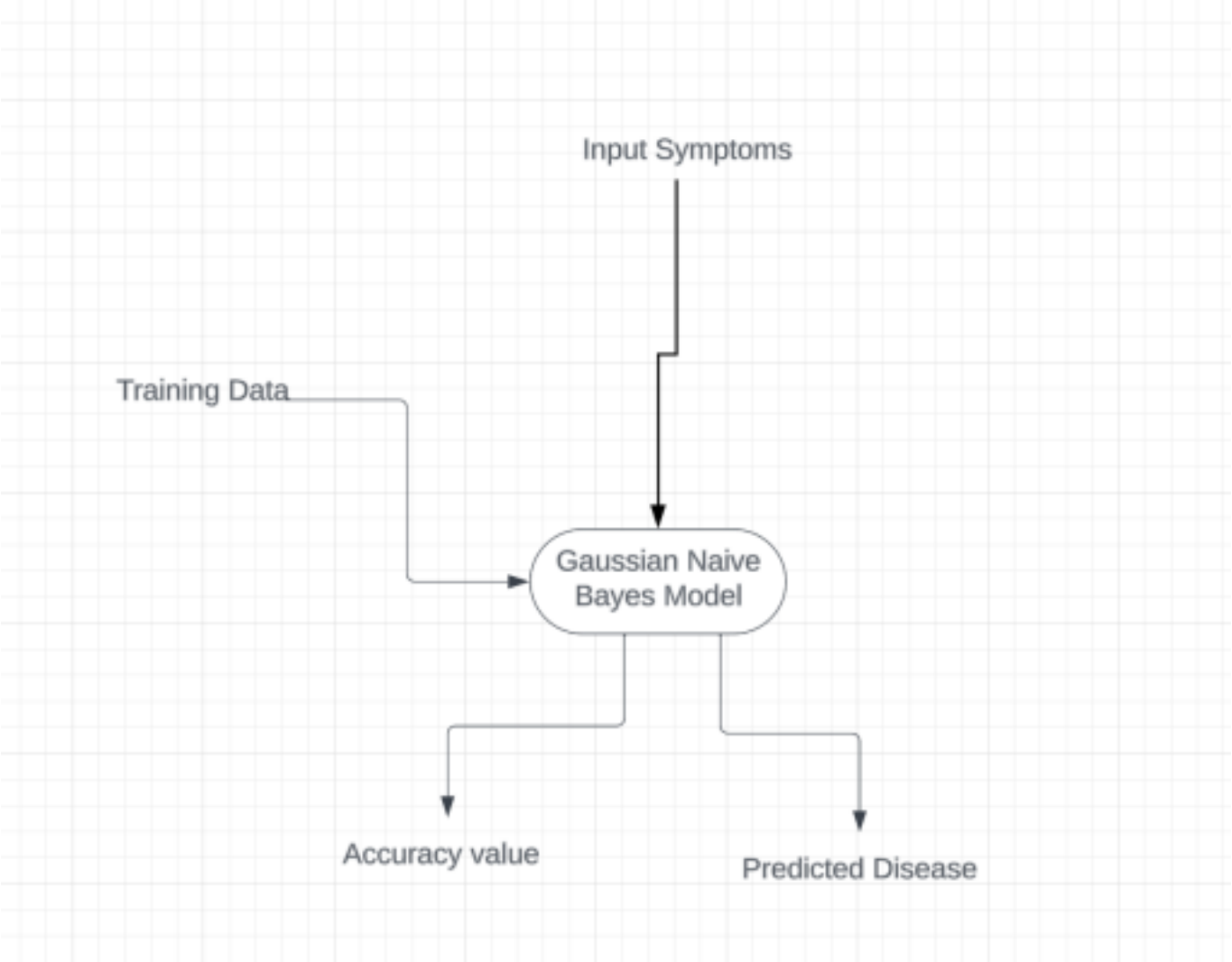
Name:Chaitanya Kakade
Batch:C21
Roll_No:2103076

Symptoms’ Prediction and Management(Model 2)



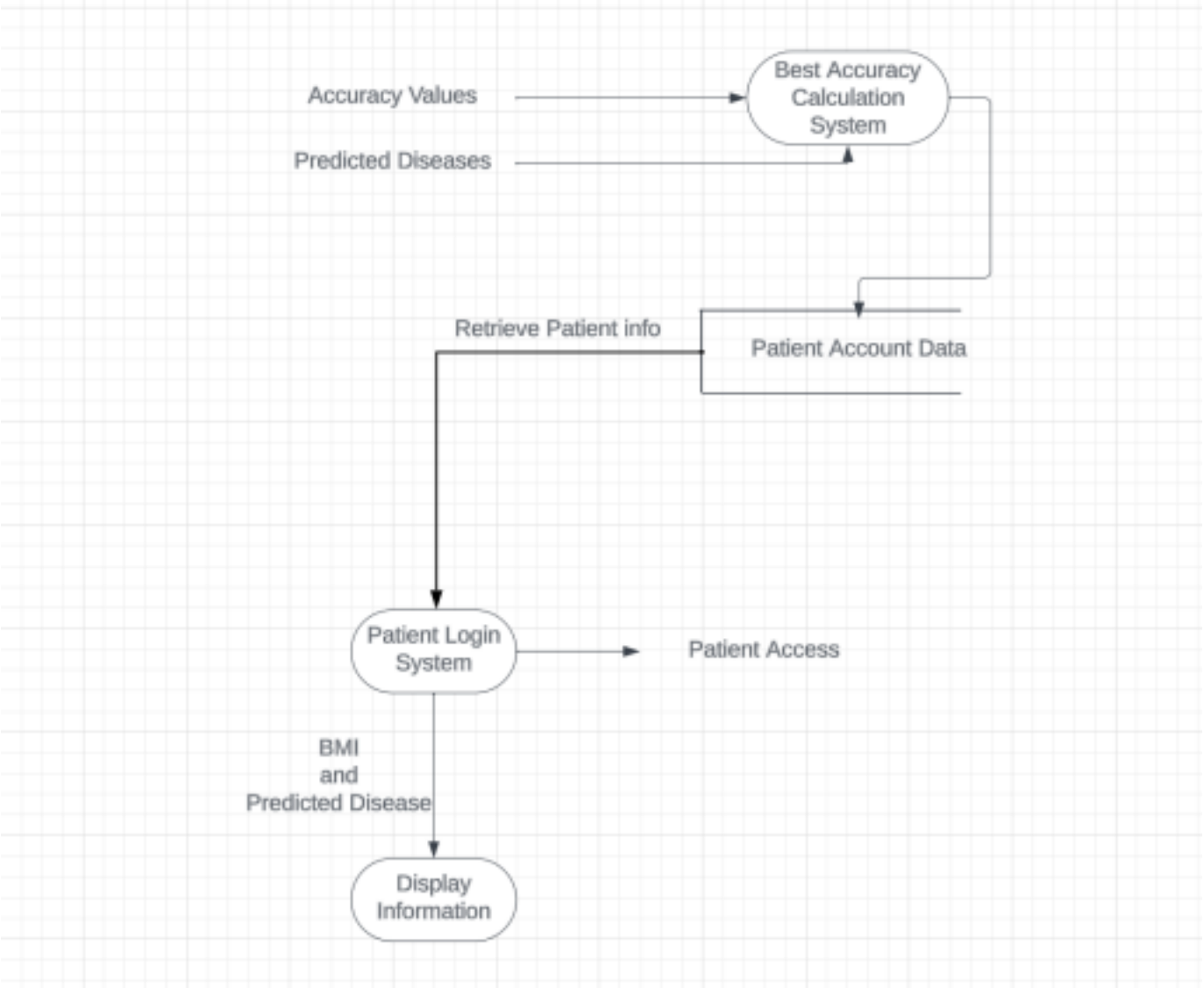
Name:Chaitanya Kakade
Batch:C21
Roll_No:2103076

Symptoms’ Prediction and Management(Model 3)



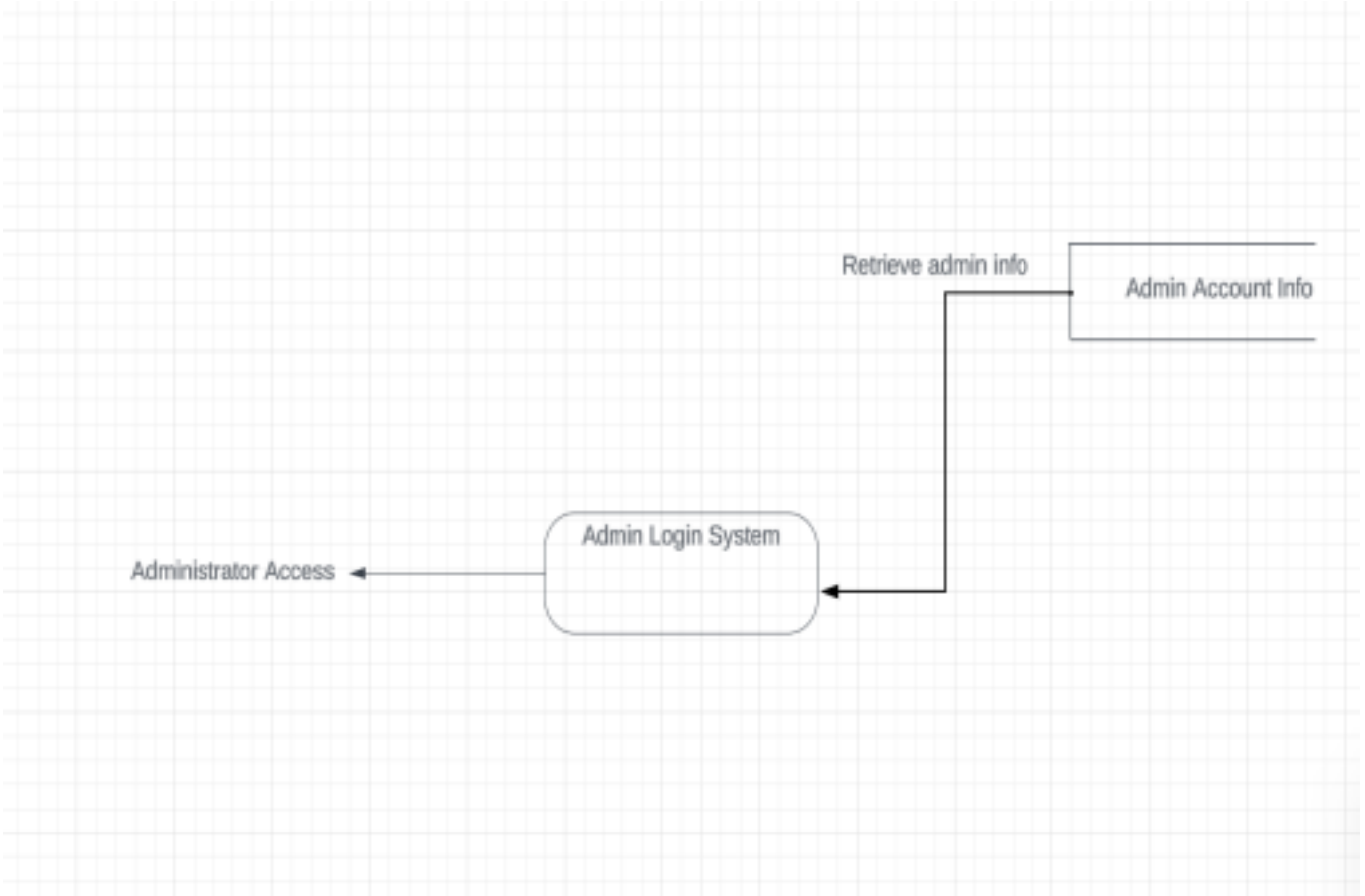
Name:Chaitanya Kakade
Batch:C21
Roll_No:2103076

Disease Information Management:



Name:Chaitanya Kakade
Batch:C21

Administration Management:



EXPERIMENT NO 5:

GROUP PROJECT TITLE: To develop Activity and State Diagram for the project

Group Members:

1)Chaitanya Kakade-2103076
2)Aman Karlupia-2103084

Name : Aman Karlupia
Roll : 2103084
Batch : C21

EXPERIMENT NO. 5

AIM: To develop Activity and State Diagram for the project

THEORY:

UML **State machine diagram** and **activity diagram** are both behavioral diagrams but have different emphases. **Activity diagram** is flow of functions without trigger (event) mechanism, **state machine** is consist of triggered states.
A state diagram (also called state machine diagram, statchart and start transition diagram) is a type of diagram behavior diagram in UML. In most OO techniques, state diagrams are drawn for a single class to show the lifetime behavior of a single object. They describe all of the possible states that a particular object (or even the entire system) can get into. State Machines give us the means to control decisions and each state is like a 'mode of operation' for the object which behaves differently depending on its state.
Activity diagrams describe activities which involve concurrency and synchronization, which are a variation of state diagrams that focuses on the flow of actions and events. They can be used for:
To model a human task (a business process, for instance).
To describe a system function that is represented by a use case.
In operation specifications, to describe the logic of an operation.

State Diagrams and Activity Diagrams are two types of Unified Modeling Language (UML) diagrams used in software engineering and system design to model different aspects of a system's behavior. Let's explore each of them:

State Diagram:
1. Purpose:
- State Diagrams, also known as State Machine Diagrams, are used to model the dynamic behavior of a system in terms of its states and transitions between states.
- They are particularly useful for modeling objects or systems that exhibit different states and respond to events that cause state transitions.

2. Components:
-State:
A state represents a condition or situation in which an object or system can exist. States are typically depicted as rounded rectangles and are labeled with names.
- ransition: A transition is a line connecting two states and represents a change from one state to another. Transitions are triggered by events or conditions.
- Event: An event is something that happens, causing a transition from one state to another. Events can be labeled on transitions.

- Actions: Actions or behaviors associated with states or transitions can be added to describe what happens when a state is entered, during its execution, or when it exits.

3. Usage: - State Diagrams are used to model complex systems where the behavior of the system can be described in terms of different states and how the system transitions between these states.
- They are commonly used in the design of real-time systems, control systems, and software components with well-defined states.

Activity Diagram:
1. Purpose:
- Activity Diagrams are used to model the workflow or procedural aspects of a system. They show the flow of activities or tasks within a system or a specific process.
- They are helpful for visualizing the steps, decision points, concurrency, and the sequence of actions in a process or workflow.

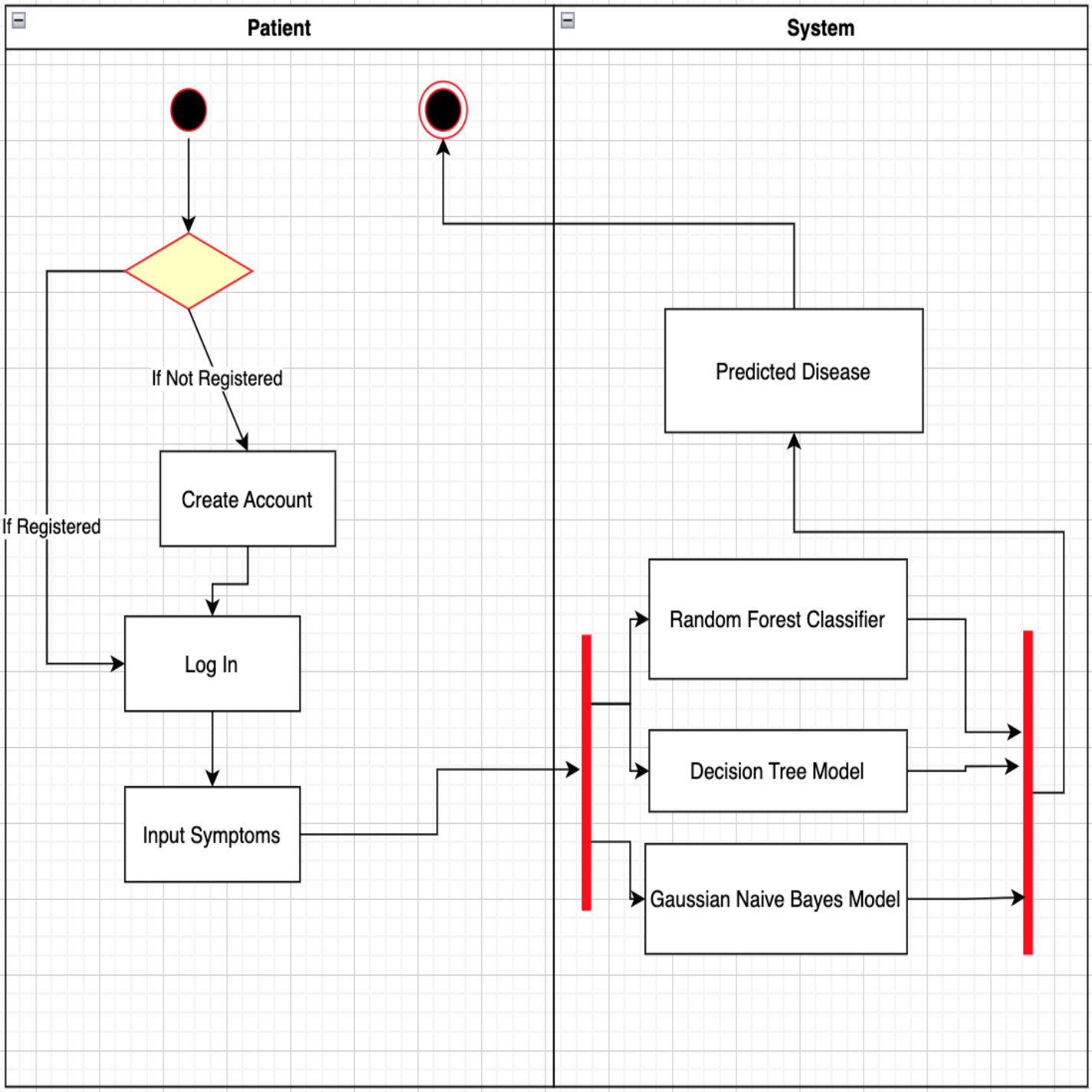
2. Components:
- Activity: An activity represents a task or action that is performed within the system or process. Activities are usually depicted as rounded rectangles and are labeled with names.
- Control Flow: Control flows represent the order in which activities are performed and are shown as arrows connecting activities.
- Decision Nodes: Decision nodes represent points in the process where a decision is made, and the flow can diverge based on conditions. They are depicted as diamonds.
- Forks and Joins: Forks indicate parallel processing, where multiple activities can occur concurrently. Joins indicate the merging of concurrent flows back into a single flow.

3. Usage:
- Activity Diagrams are used to model business processes, use cases, workflows, and any system or process where the sequence of actions and decision points is crucial to understanding the system's behavior.
- They are widely used in business process modeling and software development to depict complex workflows and business logic.

In summary, State Diagrams focus on modeling the behavior of objects or systems in terms of states and transitions, while Activity Diagrams are used to model the flow of tasks and actions within a process or workflow. Both diagrams are valuable tools in system modeling and design, and their choice depends on the specific aspect of system behavior you want to capture.

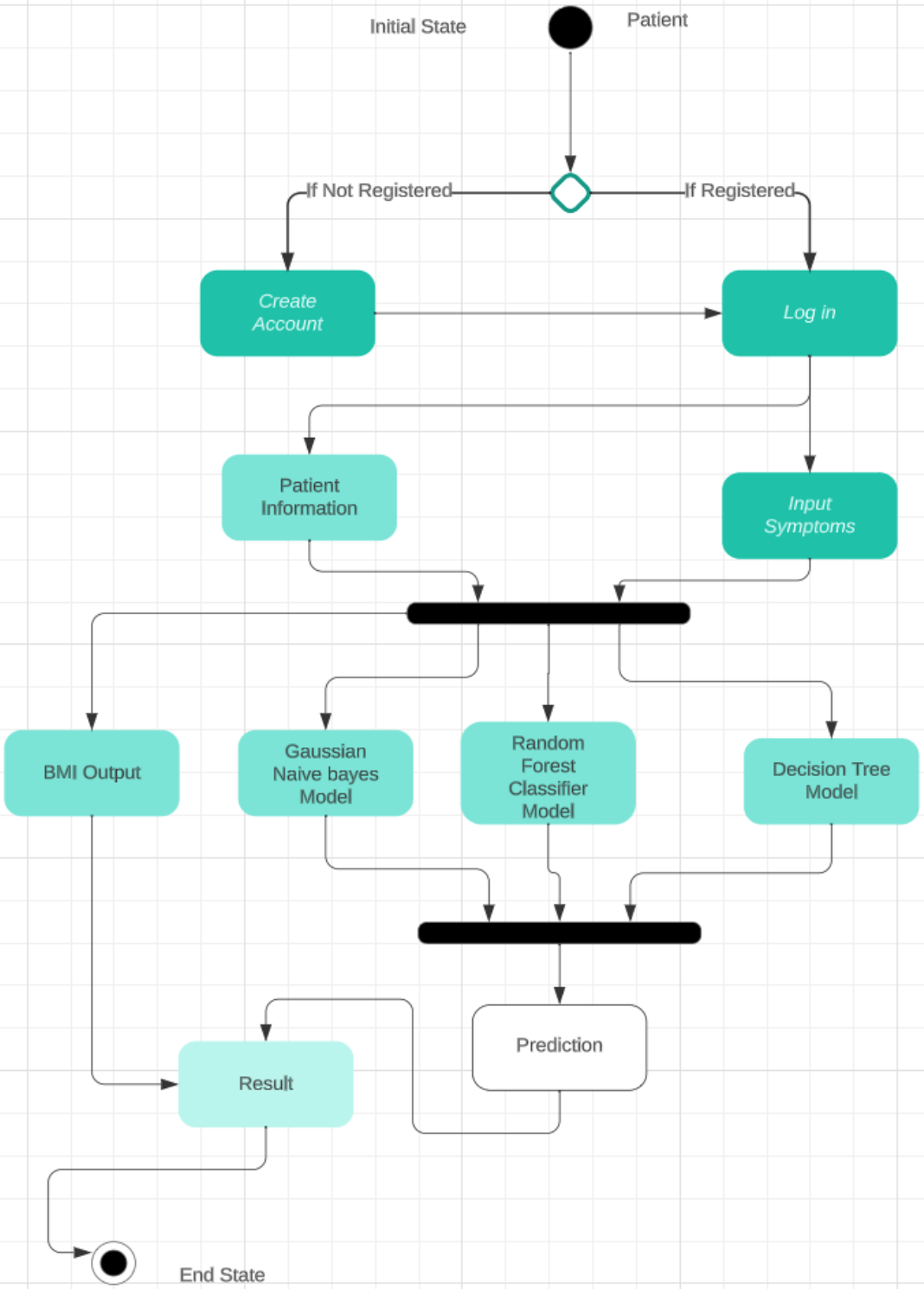
Name : Aman Karlupia
Roll : 2103084
Batch : C21

Activity Diagram



Name : Aman Karlupia
Roll : 2103084
Batch : C21

State Diagram:



EXPERIMENT NO 6:

GROUP PROJECT TITLE: To identify scenarios and develop Use Case Diagram for the project.

Group Members:

1)Chaitanya Kakade-2103076
2)Aman Karlupia-2103084

AMAN KARLUPIA
C21
2103084

EXPERIMENT NO. 6

AIM: Identify scenarios and develop Use Case Diagram for the project.

THEORY:

Use Case Diagram

A use case diagram is used to represent the dynamic behavior of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and also tells how the user handles a system.

Purpose of Use Case Diagrams

The main purpose of a use case diagram is to portray the dynamic aspect of a system. It accumulates the system's requirement, which includes both internal as well as external influences. It invokes persons, use cases, and several things that invoke the actors and elements accountable for the implementation of use case diagrams. It represents how an entity from the external environment can interact with a part of the system. Following are the purposes of a use case diagram given below:

1. It gathers the system's needs.
2. It depicts the external view of the system.
3. It recognizes the internal as well as external factors that influence the system.
4. It represents the interaction between the actors.

It is essential to analyze the whole system before starting with drawing a use case diagram, and then the system's functionalities are found. And once every single functionality is identified, they are then transformed into the use cases to be used in the use case diagram.

Use case diagram components

Actors: The users that interact with a system. An actor can be a person, an organization, or an outside system that interacts with your application or system. They must be external objects that produce or consume data.
System: A specific sequence of actions and interactions between actors and the system. A system may also be referred to as a scenario.
Goals: The end result of most use cases. A successful diagram should describe the activities and variants used to reach the goal.
ACTORS:
Actors are external entities or roles that interact with the system. Actors can represent human users, other systems, or even hardware devices. In the diagram, actors are depicted as stick figures or blocks, typically positioned on the left or top side of the diagram. Example: "User," "Administrator," "Payment Gateway."

AMAN KARLUPIA
C21
2103084

USE CASE:
A use case represents a specific piece of functionality or a discrete action that the system can perform. Use cases describe the interaction between an actor and the system to achieve a particular goal.Use cases are depicted as ovals or ellipses within the diagram, often connected to actors by lines.
Example Use Cases: "Login," "Place Order," "Generate Invoice."

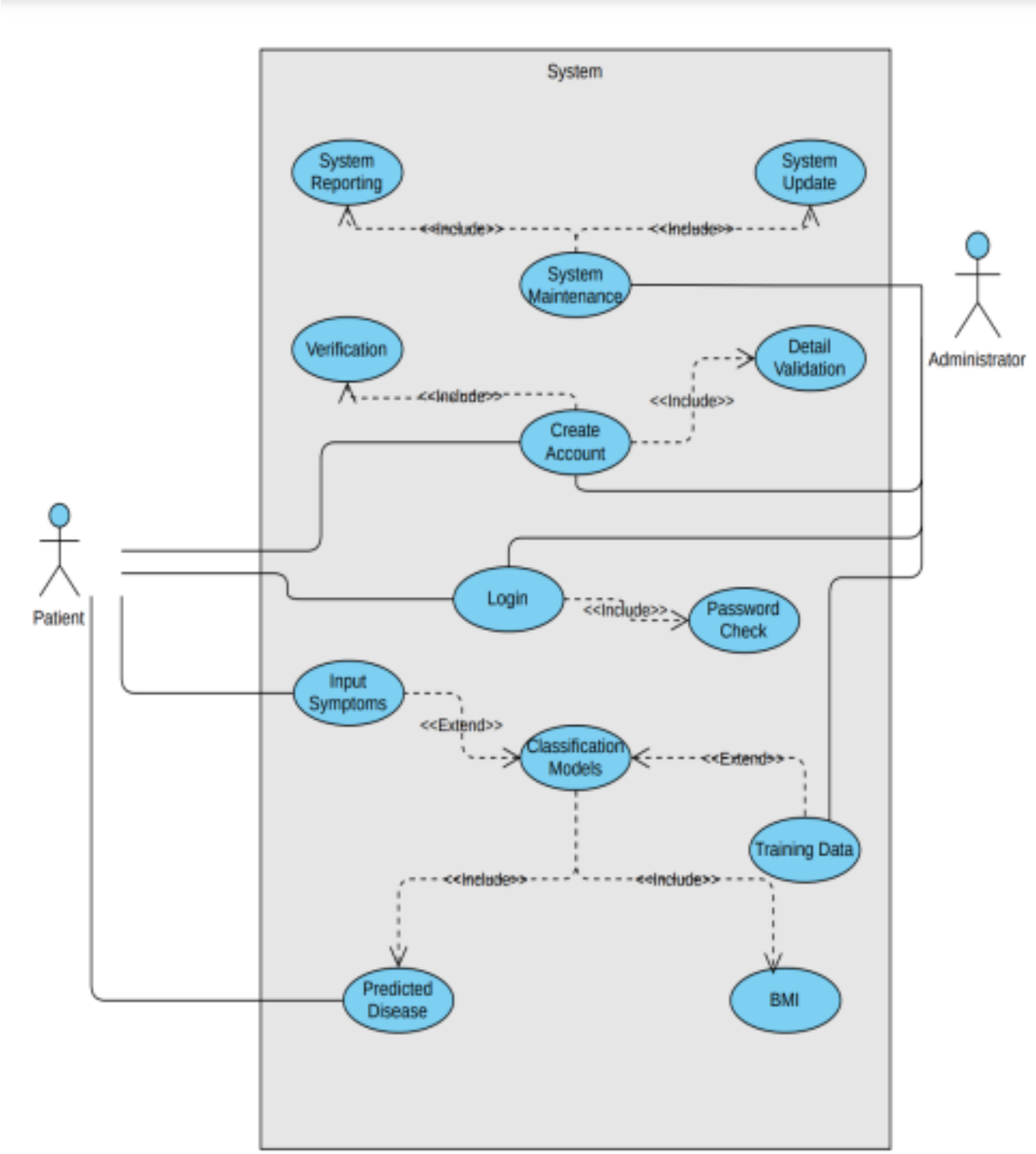
ASSOCIATIONS:
Associations represent the communication or interaction between actors and use cases. They are typically represented by lines connecting actors and use cases. Associations show that an actor is involved in or interacts with a specific use case.
Example: A line connecting an "Administrator" actor to the "Manage Users" use case.

SYSTEM:
The system boundary is a box or boundary that encloses all the use cases and actors within the diagram. It helps define the scope of the system being depicted.
While not always necessary, a system boundary can provide clarity when the system has many use cases and actors.

INCLUDE AND EXTEND RELATIONSHIP:
These are advanced concepts in Use Case Diagrams that describe how one use case can include or extend another use case under certain conditions.
"Include" relationships show that one use case includes the functionality of another. It's often used when certain steps are common to multiple use cases.
"Extend" relationships indicate that one use case can be extended with additional functionality under specific conditions.
Use Case Diagrams are a valuable tool for both technical and non-technical stakeholders to understand and communicate the functional aspects of a system. They help ensure that the system requirements align with user needs and provide a high-level overview of how the system functions in terms of user interactions.

AMAN KARLUPIA
C21
2103084

USE-CASE DIAGRAM:Disease Prediction System:



EXPERIMENT NO 7:

GROUP PROJECT TITLE: create the project schedule using Gantt chart

Group Members:

1)Chaitanya Kakade-2103076

2)Aman Karlupia-2103084

Name:Chaitanya Kakade
Batch:C21
Roll No.: 2103076

THEORY:

A Gantt chart is a widely used project management tool in software engineering and other fields. It helps visualize and plan project tasks and their dependencies over time. In software engineering, Gantt charts can be used for:

1. Task Scheduling: You can create a timeline of software development tasks, showing when each task should start and

end. This helps in allocating resources effectively.

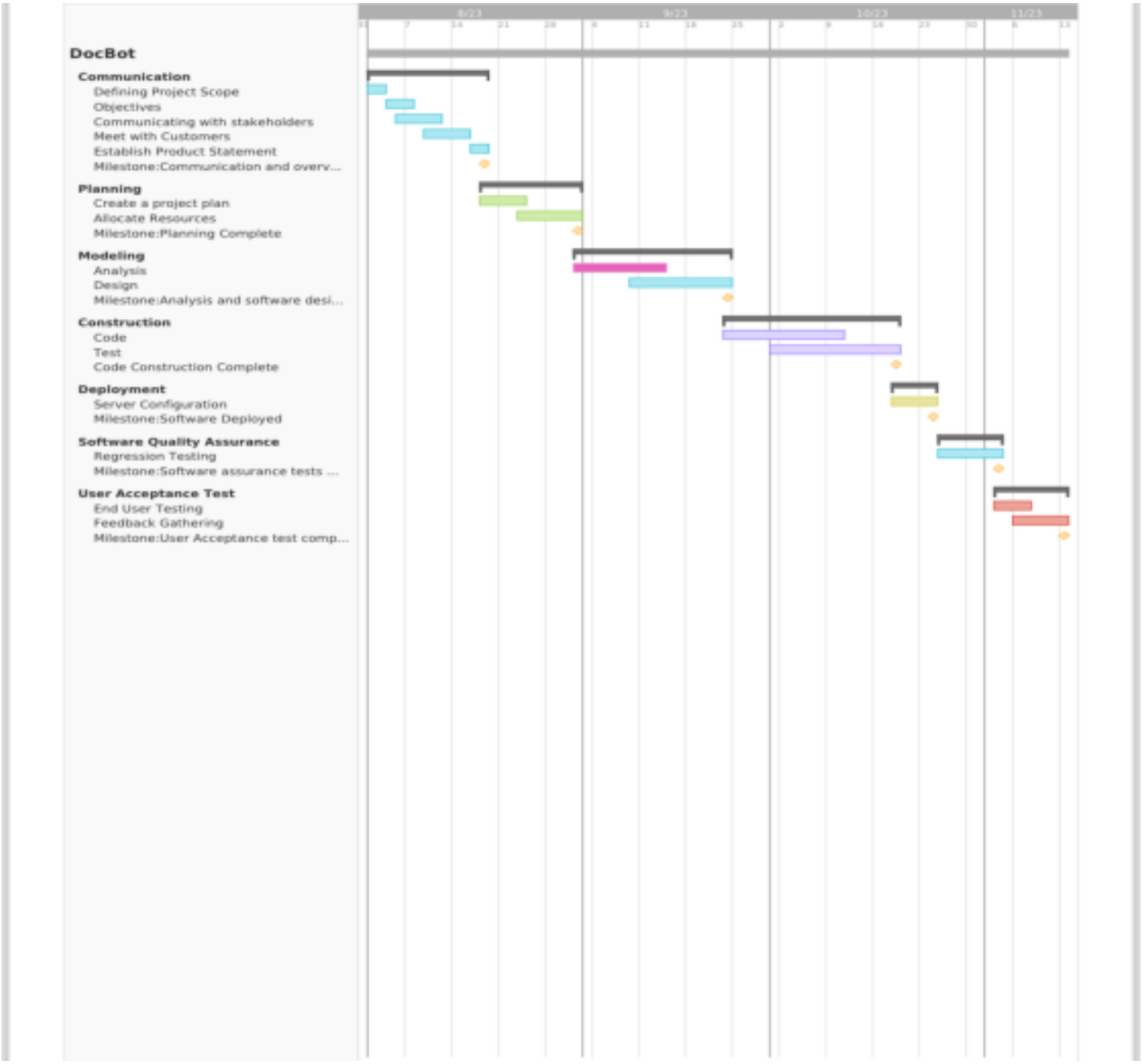
2. Dependency Management: Gantt charts show dependencies between tasks, making it clear which tasks must be completed before others can start. This is crucial in software development, where certain tasks rely on the completion of others.

3. Resource Allocation: You can assign team members or resources to specific tasks on the Gantt chart, ensuring that you have the right people working on the right tasks at the right times.

4. Progress Tracking: Gantt charts allow you to track the progress of your software project by marking completed tasks and comparing them to the planned schedule.

5. Deadline Management: They help in setting and tracking project deadlines, ensuring that the software project stays on schedule.

To create a Gantt chart for software engineering projects, you can use various project management tools and software, such as Microsoft Project, Trello, Asana, or specialized project management software that offers Gantt chart functionality. These tools make it easier to create, update, and share Gantt charts with your team, making project management more efficient and transparent.


8/23
9/23
10/23
11/23
DocBot
Communication
Defining Project Scope
Objectives
Communicating with stakeholders
Meet with Customers
Establish Product Statement
Milestone:Communication and overv...
Planning
Create a project plan
Allocate Resources
Milestone:Planning Complete
Modeling
Analysis
Design
Milestone:Analysis and software desi...
Construction
Code
Test
Code Construction Complete
Deployment
Server Configuration
Milestone:Software Deployed
Software Quality Assurance
Regression Testing
Milestone:Software assurance tests ...
User Acceptance Test
End User Testing
Feedback Gathering
Milestone:User Acceptance test comp...

EXPERIMENT NO 8:

GROUP PROJECT TITLE: To Conduct Function Point Analysis(FPA) for the project

Group Members:

1)Chaitanya Kakade-2103076
2)Aman Karlupia-2103084

AMAN KARLUPIA
C21
2103084

EXPERIMENT NO. 8

AIM:To conduct Functional Point Analysis(FPA) for Disease Prediction System.

SOLUTION:

A)Weighing Factor :- SIMPLE

Information Domain Value	Count		Weighing Factor		Total
			Simple Average	Complex	
No of User Inputs	5	*	**3** 4	6	= 15
No. of User Outputs	2	*	**4** 5	7	= 8
No. of User Inquiries	3	*	**3** 4	6	= 9
No. of Files	2	*	**7** 10	15	= 14
No. of External Files	1	*	**5** 7	10 Total Count	= 5 51

AMAN KARLUPIA
C21
2103084

A)Weighing Factor :- AVERAGE

Information Domain Value	Count		Weighing Factor		Total
No of User Inputs	5	*	Simple **Average** 3 **4**	Complex 6	= 20
No. of User Outputs	2	*	4 **5**	7	= 10
No. of User Inquiries	3	*	3 **4**	6	= 12
No. of Files	2	*	7 **10**	15	= 20
No. of External Files	1	*	5 **7**	10 Total Count	= 7 69

A)Weighing Factor :- COMPLEX

Information Domain Value	Count		Weighing Factor	Total

No of User Inputs	5	*	Simple Average 3 4	**Compl** **ex 6**	= 30
No. of User Outputs	2	*	4 5	**7**	= 14
No. of User Inquiries	3	*	3 4	**6**	= 18
No. of Files	2	*	7 10	**15**	= 30
No. of External Files	1	*	5 7	**10**	= 10
				Total Count	102

Assigning Values to the 14 Questions on a scale of 0-5

1) Data Communications 4
2) Distributed Processing 3
3) Performance Critical 4
4) Existing Operating Environment 3
5) Online Data entry 3
6) Input Transaction over multiple screens 2
7) Master files updated online 3

8) Information domain values 2 Complex 9) Internal Processing Complex 3 10) Code designed for reuse 4 11) Conversion/Installation in design 2
12) Multiple Installations 2
13)Application designed for change 3
14) Labour Cost 3

Justification:

Data communications = 4 (High score because data communication complexity can be significant in disease prediction systems that might require extensive data transfer and integration with other systems or databases.)

Distributed Processing = 3 (Moderate score as some level of distributed processing may be needed for scalability, but it's not overly complex.)

Performance Critical = 4 (High score because accurate and fast disease predictions are typically critical, requiring careful performance optimization.)

Existing Operating Environment = 3 (Moderate score since leveraging an existing environment can simplify development, but it depends on the specifics.)
Name:Chaitanya Kakade
Batch:C21
Roll_No:2103076

Online Data entry = 3 (Moderate score as online data entry is common but not overly complex on its own.)

Input transaction over multiple screens = 2 (Lower score because input transactions, even across screens, are usually straightforward.)

Master Files updated Online = 3 (Moderate score as online updates of master files are common but require careful handling.)

Information domain values Complex = 2 (Lower score because handling complex information domain values is common but not highly complex.)

Internal Processing Complex = 3 (Moderate score as internal processing can vary

in complexity, depending on specific algorithms used.)

Code designed for reuse = 4 (High score because designing for code reuse is beneficial but requires extra effort and complexity.)

Conversion/Installation in design = 2 (Lower score because conversion/installation, while important, is typically manageable.)

Multiple Installations = 2 (Lower score as handling multiple installations is a common task.)

Application designed for change = 3 (Moderate score because designing for change is important but not overly complex.)

Labour Cost = 3 (Moderate score as labor costs can have a moderate impact on project complexity, depending on team size and skill.)

Total Value Adjustment Factor => ∑Fi = 41

Function point values:

	Calculation Function Point
Simple	51*[0.65+0.01*∑Fi] = 54.06
Average	69*[0.65+0.01*∑Fi] = 73.14
Complex	102*[0.65+0.01*∑Fi] =108.12

Cost for each Method:

(Assuming 1 FP = 1450 units)

Method Cost
Simple 78,387 (54.06*1450)
Average 1,06,053 (73.14*1450)
Complex 1,56,774 (108.12*1450)

CONCLUSION:

The function point values and cost for building the Disease Prediction System were calculated as shown above.

AMAN KARLUPIA
C21
2103084

EXPERIMENT NO.9

AIM: Application of COCOMO model for cost estimation of the project

THEORY:

Boehm proposed COCOMO (Constructive Cost Estimation Model) in 1981.COCOMO is one of the most generally used software estimation models in the world. COCOMO predicts the efforts and schedule of a software product based on the size of the software.

In COCOMO, projects are categorised into three types:

1. Organic
2. Semi Detached
3. Embedded

Basic COCOMO 1

Software Project	a1	a2 b1 b2
Organic	2.4	1.05 2.5 0.38
Semi-detached	3.0	1.12 2.5 0.35
Embedded	3.6	1.20 2.5 0.32

Formulae:

1) Estimation of Effort:

-Organic: EFFORT = $2.4*(KLOC)^{1.5}$ PM

- Semi-Detached: EFFORT = $3.0*(KLOC)^{1.12}$ PM

- Embedded: EFFORT = $3.6*(KLOC)^{01.20}$ PM

2) Estimation of Development Time:

-Organic: Tdev = $2.5*(Effort)^{0.38}$ Months

- Semi-Detached: Tdev = $2.5*(Effort)^{0.35}$ Months

- Embedded: Tdev = $2.5*(Effort)^{0.32}$ Months

3) Estimation of Productivity:

Productivity = ��������/����������

4) Estimation of Average Staff Size(SS):

Average Staff size = ����������/�������������������
��������

Question:

For developing a Disease Prediction System,we have estimated that our project will have 12 KLOC.Find the effort,development time,productivity and average staff size for each of the three development modes:Organic,Semidetached and embedded for BASIC COCOMO 1.

SOLUTION:

Step 1:Organic:

Using the above mentioned formulae,we calculate the estimates as follows: Effort = $2.4*(12)^{1.5}$ PM = **99.76 PM**

Tdev = $2.5*(\text{Effort})^{0.38}$ Months = **14.37 months**

Productivity = ��������/���������� = 12/99.76 = **0.1202 KLOC/PM**

Average Staff size = ����������/������������������ ��������= 99.76/14.37 = 6.9422 = **7 person**

Step 2:Semi-Detached:

Using the above mentioned formulae,we calculate the estimates as follows: Effort = $3.0*(\text{KLOC})^{1.12}$ PM = **48.50 PM**

Tdev = $2.5*(\text{Effort})^{0.38}$ Months = **10.927 months**

Productivity = ��������/���������� = 12/48.50 = **0.247 KLOC/PM**

Average Staff size = ����������/������������������ ��������= 48.50/10.927 =4.4385 = **4 person**

Name:Chaitanya Kakade
Batch:C21
Roll No. 2103076

AMAN KARLUPIA
C21
2103084

Step 3: Embedded

Using the above mentioned formulae,we calculate the estimates as follows:

Effort =3.6*(KLOC)$^{01.20}$ PM = **71.01 PM**

Tdev =2.5*(Effort)$^{0.32}$ Months = **9.78 months**

Productivity = ********** / ********** = 12/71.01= **0.168 KLOC/PM**

Average Staff size = ********** / ********** ********** = 71.01/9.78 =7.26 = **7 person**

Intermediate COCOMO 1

Software Project	a1	a2 b1 b2
Organic	3.2	1.05 2.5 0.38
Semi-detached	3.0	1.12 2.5 0.35
Embedded	2.8	1.20 2.5 0.32

Intermediate COCOMO equation:

$E=a_i (KLOC) b_i*EAF$
$D=c_i (E)d_i$

Cost Drivers	RATINGS					
	Very low	Low	Nominal	High	Very High	Extra High
Product Attributes						
RELY	0.75	0.88	1.00	1.15	1.40	--
DATA	--	0.94	1.00	1.08	1.16	--
CPLX	0.70	0.85	1.00	1.15	1.30	1.65
Computer Attributes						
TIME	--	--	1.00	1.11	1.30	1.66
STOR	--	--	1.00	1.06	1.21	1.56
VIRT	--	0.87	1.00	1.15	1.30	--
TURN	--	0.87	1.00	1.07	1.15	--

Cost Drivers	RATINGS					
	Very low	Low	Nominal	High	Very high	Extra high
Personnel Attributes						
ACAP	1.46	1.19	1.00	0.86	0.71	--
AEXP	1.29	1.13	1.00	0.91	0.82	--
PCAP	1.42	1.17	1.00	0.86	0.70	--
VEXP	1.21	1.10	1.00	0.90	--	--
LEXP	1.14	1.07	1.00	0.95	--	--
Project Attributes						
MODP	1.24	1.10	1.00	0.91	0.82	--
TOOL	1.24	1.10	1.00	0.91	0.83	--
SCED	1.23	1.08	1.00	1.04	1.10	--

Formulae:

5) Estimation of Effort:

-Organic: EFFORT = $3.2*(KLOC)^{1.05}$ PM

- Semi-Detached: EFFORT = $3.0*(KLOC)^{1.12}$ PM

- Embedded: EFFORT = $2.8*(KLOC)^{1.20}$ PM

6) Estimation of Development Time:

-Organic: Tdev = $2.5*(Effort)^{0.38}$ Months

- Semi-Detached: Tdev = $2.5*(Effort)^{0.35}$ Months

- Embedded: Tdev = $2.5*(Effort)^{0.32}$ Months

7) Estimation of Productivity:

Productivity = ��������/����������

8) Estimation of Average Staff Size(SS):

Average Staff size = ����������/������������

Question:

For developing a Disease Prediction System,we have estimated that our project will have 12 KLOC.Find the effort,development time,productivity and average staff size for each of the three development modes:Organic,Semidetached and embedded for Intermediate COCOMO 1.

SOLUTION:

Let the Cost Drivers be set to following parameters:

Product Cost Drivers = High

Computer Cost Drivers = Nominal

Personnel Cost Drivers = Low

Project Cost Drivers = High

Product cost drivers(from the table) = 1.15*1.08*1.15 = 1.43(for HIGH) Computer cost drivers(from the table) = 1.00(for Nominal) Personnel cost drivers(from the table) = 1.19*1.13*1.17*1.10*1.07 = 1.85(for low) Project cost drivers(from the table) = 0.91*0.91*1.04 = 0.86

Total PRODUCT of all cost drivers = (1.43*1.00*1.85*0.86) = 2.28

Step 1 (ORGANIC):

Using the above mentioned formulae,we calculate the estimates as follows: Effort = $3.4*(12)^{1.05}$ *2.28PM = **105.330 PM**

Tdev = $2.5*(\text{Effort})^{0.38}$ Months = **14.67 months**

Productivity = ********** / ********** = 12/105.330 = **0.1139 KLOC/PM**

Average Staff size = ********** / ********** ********** = 105.330/14.67 = 7.17 = **7 people.**

Step 2:Semi-Detached:

Using the above mentioned formulae,we calculate the estimates as follows: Effort = $3.0*(12)^{1.12}$ *2.28PM = **110.59 PM**

Tdev = $2.5*(\text{Effort})^{0.38}$ Months = **17.93 months**

Productivity = ********** / ********** = 12/110.59 = **0.108 KLOC/PM**

Average Staff size = ********** / ********** ********** = 110.59/17.93 =6.16= **6 person**

Step 3: Embedded

Using the above mentioned formulae,we calculate the estimates as follows: Effort =$2.8*(12)^{1.20}$ *2.28PM = **125.92 PM**

Tdev =$2.5*(\text{Effort})^{0.32}$ Months = **11.74 months**

Productivity = ********** / ********** = 12/125.92= **0.095 KLOC/PM**

Average Staff size = ********** / ********** ********** = 125.92/11.74 =10.72= **11 person**

COCOMO 2:

FORMULAE:

1)NOP = (Object points)*[(100-%reuse)/100]

2)Effort = NOP/PROD

Complexity Weights for COCOMO 2 model are:

Object Type	Simple	Medium Difficult
Screen	1	2 3
Report	2	5 8
3 GL component		10

Productivity Table:

Developers' Experience/capability	Very Low	Nominal	High	Very High
Environment maturity/capability	Very Low	Nominal	High	Very High
Productivity	4	7 13	25	50

For screens

No of Views	Tot<4 (<2 servers,<3 clients)	Tot<8(2-3 servers,3-5 clients) Tot = 8+ (>3servers,>5 clients)
<3	Simple	Simple Medium
3-7	Simple	Medium Difficult
>8	Medium	Difficult Difficult

For Reports

No of Sections	Tot<4 (<2 servers,<3 clients)	Tot<8(2-3 servers,3-5 clients) Tot = 8+ (>3servers,>5 clients)
0-1	Simple	Simple Medium

2-3	Simple	Medium Difficult
4+	Medium	Difficult Difficult

QUESTION:
Using COCOMO 2 Model,estimate the effort required to build software for a simple disease prediction system that produces 5 screens(2 views each,7 data tables for three servers and 4 clients),6 reports(2 sections from 7 data tables for 2 servers and 3 clients),and will require approximately 80% of new software components.
Make use of an application composition model with object points and calculate NOP,EFFORT.

SOLUTION:
Step 1:
No of screens = 5
No of reports = 6

Step 2 :(For screen):
No. of views = 2
No of data tables = 7
No of servers = 3
No of clients = 4
By using this info,Complexity level for each screen = **simple**

Steep 3(For report):
No. of sections = 2
No of data tables = 7
No of servers = 2
No of clients = 3
By using this info,Complexity level for each report = **difficult**

Given:

Object	Count	Complexity Weight Factor Total Objects
Screens	5	**Simple** 1 5

Name:Chaitanya Kakade
Batch:C21
Roll No. 2103076

Reports	6	**Difficult** 8 48
3 GL Components	0	N/A 0

Total Object Points 53

Computing NOP(New Object Point):

Since 80% of the components are required,it means that 20% can be reused.

Computing NOP as follows:
NOP = (Object points)*[(100-%reuse)/100]

NOP = (53)*[100-20]/100
NOP = 53*80/100

NOP = 42.4

Computing Effort:

Since Productivity is average, we can assume PROD = 13.

Hence,
Effort = NOP/PROD
= 42.4/13
=3.26 person Months

Effort = 3.26 person Months

EXPERIMENT NO 10:

GROUP PROJECT TITLE: To develop a Risk Mitigation,Monitoring and Management plan(RMMM) for the project.

Group Members:

1)Chaitanya Kakade-2103076

2)Aman Karlupia-2103084

AIM: To develop a Risk Mitigation,Monitoring and Management plan(RMMM) for the project.

THEORY:

"Risk" refers to a situation that could result in a loss or jeopardise the project's progress but has not yet occurred.
Risk management is the process of identifying risks and developing plans to mitigate their influence on the project. The primary goal of risk management is to avert disasters or significant losses.
The Risk Mitigation, Monitoring, and Management(RMMM) plan is a document that outlines all risk analysis activities. This document is sometimes included in the overall project plan by the project manager. Although a precise RMMM strategy is not always established, each risk can be described using a risk information sheet.

Risk Mitigation, Monitoring and Management(RMMM) Plan

In most cases, a risk management approach can be found in the software project plan. This can be broken down into three sections: risk mitigation, monitoring, and management (RMMM). All work is done as part of the risk analysis in this strategy. The project manager typically uses this RMMM plan as part of the overall project plan.
Some development teams use a Risk Information Sheet(RIS) to document risk. For faster information handling, such as creation, priority sorting, searching, and other analyses, this RIS is controlled by a database system. Risk mitigation and monitoring will begin after the RMMM is documented and the project is launched.

Risk Mitigation

Risk Mitigation is a technique for avoiding risks (Risk Avoidance).
The following are steps to take to reduce the risks:

- Identifying the risk.
- Getting rid of the causes that lead to the production of risk.
- Controlling the relevant documents regularly.
- Conducting timely reviews to accelerate the process.

Risk Monitoring

Risk monitoring is an activity used to track a project's progress.
The following are the critical goals of the task.

- To see if the risks that were anticipated actually happen.
- To verify that the risk aversion steps defined for risk are adequately implemented.
- To gather information for future risk assessments.
- To determine which risks generate which problems throughout the project.

Risk Management and Planning

Risk management and planning are based on the assumption that the mitigation effort failed and the risk has become a reality. When a risk becomes a reality and produces serious problems, the project manager is in charge of this responsibility. It is easier to manage risks if the project manager successfully implements project mitigation to eliminate risks. This demonstrates how a manager will respond to each risk. The risk register is the key objective of the risk management plan. This risk register identifies and prioritizes potential dangers to a software project.

RMMM Table on Disease Prediction System

Risk
User Authenticati on and Security
Model Accuracy
Algorithm Bias and Fairness
Scalability (Larger user base)

User Engagement

Risk Type Probability

Impact Security Risk 30% 3

Technical Risk 20% 4

Technical Risk 60% 3

Product size 60% 3

User Risk 60% 4

Risk Information Sheet		
Risk Id:1234	**Date:1/09/2023**	**Prob:30% Impact:3**

Description:

- Unauthorised access to user accounts or sensitive health data could easily compromise the system security.

Mitigation

- Implement multi-factor authentication(MFA) to enhance user login security. ● Conducting Regular security audits to identify potential vulnerabilities. ● Using stronger encryption module to encrypt the data during transmission and storage
- Implementing stricter access control to restrict user data to authorised personnel only.

Monitoring:

- Employ intrusion detection systems (IDS) and intrusion prevention systems (IPS) to continuously monitor network traffic for suspicious activities. ● Implement Security Information and Event Management (SIEM) solutions to analyze and correlate security events.

Management:

- Evaluate and monitor the security practices of third-party vendors and partners who have access to your systems or data.
- Implement encryption protocols for sensitive data at rest and in transit to protect against unauthorized access.

Current Status:

10/09/2023 Monitoring steps initiated

Originator: Chaitanya Kakade	**Assigned:Aman Karlupia**

Risk Information Sheet	
Risk Id:2234	**Date:20/09/2023 Prob:20% Impact:4**

Description:

- The accuracy of the disease prediction model used, could get compromised over time due to changes in the combination of symptoms and addition of new diseases.

Mitigation:

- Monitor the Models Accuracy regularly.
- Track User feedback to determine the changes
- Collect high-quality, clean, and well-labeled data to reduce noise and bias in the training dataset.
- Implement data validation and preprocessing techniques to ensure data quality.

Monitoring:

- Continuously monitor model performance using appropriate metrics (e.g., accuracy, precision, recall, F1-score) on a validation dataset.
- Set up automated performance tracking and alerting systems for real-time monitoring.
- Utilize anomaly detection techniques to identify unusual model behavior, such as sudden drops in accuracy or unexpected errors.

Management:

- Fine-tune hyperparameters or update the model architecture to improve accuracy when it degrades due to changing data patterns.
- Implement a feedback loop where model predictions are continuously evaluated and compared to actual outcomes.
- Adjust the model's predictions based on real-world results.

Current Status:

24/09/2023 Monitoring steps initiated.

Originator:Chaitanya Kakade Assigned:Aman Karlupia

Risk Information Sheet	
Risk Id:2332	**Date:30/09/2023 Prob:60% Impact:3**

Description:

Addressing algorithm bias and fairness is crucial in machine learning and data science to ensure that models are not discriminatory and provide equitable results.

Mitigation:

- Choose machine learning algorithms that are less sensitive to bias or have built-in fairness considerations.
- Implement pre-processing or in-processing bias mitigation techniques when necessary.

Monitoring:

- Implement concept drift detection to identify shifts in the data distribution that may introduce bias.
- Regularly assess model performance and fairness across different subpopulations.
- Encourage users and stakeholders to provide feedback on model outputs to identify instances of bias or unfair treatment.

Management:

- Fine-tune algorithm hyperparameters or model architecture to improve fairness while maintaining acceptable performance.
- When bias is detected, take corrective action by retraining models with updated and debiased data.
- Explore reweighting or re-sampling techniques to reduce disparate impact.

Current Status:

1/10/2023 Mitigation steps initiated

Originator:Chaitanya Kakade Assigned: Aman Karlupia

AMAN KARLUPIA
C21
2103084

Risk Information Sheet		
Risk Id:4903	**Date:7/10/2023**	**Prob:60% Impact: 3**

Description:
Identify potential scalability risks such as increased system load,performance bottlenecks, and resource limitations.
Consider the possible consequences of not being able to scale effectively, such as system downtime, slow response times, and user dissatisfaction.

Mitigation:
- Adopt cloud-based solutions that offer auto-scaling capabilities to handle fluctuating user loads.
- Regularly conduct load testing and stress testing to identify scalability issues early in the development process.

Monitoring:
- Implement performance monitoring tools to keep a close eye on system metrics such as CPU usage, memory, and network bandwidth.
- Set up alerts to notify the operations team when performance thresholds are exceeded.
- Continuously track user growth and system usage patterns to anticipate future scaling needs.

Management:
- Capacity Planning: Allocate resources and capacity based on the observed system usage patterns and projected user growth.
- Scaling Strategies: When the system nears capacity, follow predefined scaling strategies, such as adding more servers, increasing database capacity, or optimizing code.
- Failover and Redundancy: Implement redundancy and failover mechanisms to ensure system availability and minimize downtime.

Current Status:
10/10/2023 Monitring steps initiated

Originator:Chaitanya Kakade	**Assigned:Aman Karlupia**

Risk Information Sheet	
Risk Id:7878	**Date:14/10/2023 Prob:60 Impact:4**

Description:
In this phase, potential user engagement risks are identified and documented. This involves recognizing challenges such as declining user activity, poor retention, or other issues that can negatively impact user engagement. The goal is to be aware of the risks that the project or product may face in this regard.

Mitigation:
- User-Centric Design: Ensure that the product or service is designed with the user in mind, incorporating user feedback and usability best practices.
- Continuous Improvement: Regularly update and enhance the product or service to maintain user interest and relevance.
- Personalization: Implement personalization features that tailor user experiences to individual preferences and needs.

Monitoring:
- Implement user analytics and tracking tools to monitor user engagement metrics, such as active users, session duration, and conversion rates. ● Conduct user surveys and feedback collection to gain insights into user satisfaction and identify areas for improvement.

Management:
- Content Strategy: Continuously update and optimize the content, features, or services offered to maintain user interest.
- User Support: Provide effective user support and assistance to address user concerns and issues promptly.
- A/B Testing: Implement A/B testing to assess the impact of changes on user engagement and user behavior.

Current Status:
17/10/2023-Monitoring initiated

Originator:Chaitanya Kakade Assigned:Aman Karlupia

Risk	Risk Type	Probability Impact RMMM
Data Privacy	Security risk	70% 5 ● While constructing the website try to leave loopholes for hacking as low as possible. ● Timely checkup for loopholes.
Model Reliability	Technical risk	20% 3 ● Implement real-time monitoring to assess the model's performance and detect anomalies, and define clear thresholds that trigger actions, such as retraining or alerts. ● Schedule periodic model updates and refinements, accompanied by validation checks, to keep the model's performance consistent and aligned with evolving data patterns and user needs.
Website is not user friendly	Design and usability	10% 2 ● Continuously update the features of the website. ● Time to time enhances the user interface. ● Conduct some surveys to get feedback about the website and work on it.
Limited Generalisation	Technical risk	30% 2 ● Ensure the model is trained on a diverse dataset covering a wide range of scenarios. ● Continuously update the model using fresh data to maintain its adaptability to changing real-world patterns.
Sudden resignation of experienced technician	Staff size and experience	30% 2 ● Hiring a more experienced technician. ● Proper training to other technicians who can handle the work. ● Take care of employee needs and benefits.

AMAN KARLUPIA
C21
2103084

Risk Information Sheet	
Risk Id:1224	**Date:1/09/2023 Prob:**70% **Impact:**5

Description:

- Risk of unauthorised access or data breaches compromising personal health information

Mitigation/Monitoring:

- Collect and store only necessary health data, encrypt it to prevent unauthorised access.
- Integrate privacy considerations into the systems design, establish data retention policies, and train personnel in data privacy best practices to proactively protect sensitive data.

Management:

- Identify potential data privacy risks specific to your system, such as unauthorised access, data breaches.
- Evaluate the likelihood and potential impact of each identified risk. Determine which risk poses the greatest threat.

Current Status:
7/09/2023 : Mitigation steps initiated

Originator:Aman Assigned:Chaitanya

Risk Information Sheet	
Risk Id:2241	**Date:10/09/23 Prob:**20% **Impact:**3

Description:

- Risk for the trustworthiness of the model's prediction and consistency.

Mitigation/Monitoring:

- Implement robust data monitoring and preprocessing techniques to detect and address data drift, anomalies, and errors. Ensure ethical and representative data sampling to reduce bias in training data.
- Regularly update and retrain the model using fresh data, incorporate user feedback, and enhance model interpretability to improve its trustworthiness and consistency.
- Proactively identify and correct bias in the model, addressing fairness issues, and avoiding skewed predictions, while adhering to ethical guidelines and regulatory requirements.

Management:

- Establish ongoing monitoring and maintenance protocols to ensure the model's reliability remains consistent over time. This includes regular assessments, data quality checks, and responsiveness to changes in data distribution.
- Foster collaboration between data scientists, domain experts, and stakeholders to address ethical considerations, interpretability, and feedback loops for continuous improvement.
- Ensure regulatory compliance, transparent communication with stakeholders, and alignment with best practices to build trust in the model's reliability.

Current Status:
15/09/23: Mitigation steps initiated

Originator:Aman Assigned:Chaitanya

Risk Information Sheet		
Risk Id:2341	**Date:19/09/23**	**Prob:**10% **Impact:**2

Description:

- Risk of reduced user engagement, increased bounce rates, lower customer satisfaction leading to decreased traffic and revenue.

Mitigation/Monitoring:

- Prioritise user-centred design principles by involving users in the website design process, conducting usability testing, and gathering feedback to create an intuitive and user-friendly interface.
- Continuously monitor and assess website usability and performance, making necessary improvements based on user feedback and data analytics.
- Invest in employee training and resources to ensure that the website development and maintenance team follows best practices for user-friendly design and adherence to accessibility standards.

Management:

- Prioritise user needs by involving UX/UI experts, conducting regular user testing, and maintaining a feedback loop to continuously enhance the website's usability.
- Implement an agile development process, allowing for rapid iterations and updates based on user feedback, data analytics, and emerging best practices.
- Establish robust quality assurance processes to ensure website usability, accessibility, and compliance with design standards throughout the development and maintenance life cycle.

Current Status:
23/09/23: Mitigation steps initiated.

Originator:Aman	**Assigned:Chaitanya**

Risk Information Sheet		
Risk Id:4351	**Date:27/09/23**	**Prob:**30% **Impact:**2
Description: ● Risk related to inability to make accurate predictions or inferences beyond the specific data it was trained on, resulting in reduced reliability in real-world scenarios.		
Mitigation/Monitoring: ● Train the model on a diverse and representative dataset that covers a wide range of scenarios to improve its ability to generalise to real-world cases. ● Continuously update and retrain the model with fresh data to ensure its adaptability to changing patterns and trends. ● Enhance the model's feature engineering to extract more meaningful and relevant information from the data, allowing it to generalise better to a broader range of situations.		
Management: ● Develop a data acquisition and maintenance strategy to ensure data diversity, quality, and relevance over time. ● Continuously monitor the model's performance and assess its ability to generalise to new data patterns. ● Implement a cycle of regular model updates, fine-tuning, and feature engineering to enhance generalisation and adapt to evolving real-world scenarios.		
Current Status: **1/10/23:** Mitigation steps initiated.		
Originator:Aman		**Assigned:Chaitanya**

Risk Information Sheet		
Risk Id:2353	**Date:5/10/23**	**Prob:**30% **Impact:**2
Description: ● Risk of the resignation of an experienced technician can disrupt workflows, impact project timelines, and necessitate immediate recruitment and knowledge transfer.		
Mitigation/Monitoring: ● Ensure multiple team members have knowledge of critical tasks and responsibilities to reduce dependence on a single individual. ● Establish a succession plan with a pool of potential replacements and a clear onboarding process to ease the transition. ● Encourage comprehensive documentation of processes, procedures, and technical knowledge to facilitate a smooth handover to a new hire or existing team members.		
Management: ● Quickly assess the situation, address any urgent issues, and notify relevant stakeholders. ● Plan and facilitate a structured knowledge transfer process to minimise the impact on ongoing projects and operations. ● Initiate the recruitment process for a replacement and consider interim support options to bridge the gap during the transition.		
Current Status: **7/10/23:** Mitigation steps initiated.		
Originator:Aman		**Assigned:Chaitanya**

EXPERIMENT NO 11:

GROUP PROJECT TITLE: Case Study:Github for Version Control

Group Members:

1)Chaitanya Kakade-2103076

2)Aman Karlupia-2103084

Name:Chaitanya Kakade
Roll No.:2103076
Batch:C21

THEORY:

GitHub is a web-based platform and service that plays a pivotal role in modern software development. It serves as a hub for code collaboration, version control, and project management. Developers use GitHub to host, share, and collaborate on software projects, making it an essential tool for open-source and private development. It is built around the Git version control system and offers features like code repositories, issue tracking, collaboration tools, and continuous integration, enabling teams to work efficiently and transparently. GitHub facilitates code contributions, peer reviews, and collaboration among developers, making it a central platform for the global software development community. It has gained widespread adoption and is instrumental in enabling distributed and collaborative software development across the globe.
Certainly, here is a more detailed description of GitHub:

GitHub is a web-based platform and service that provides a comprehensive set of tools and features for software development and project management. It is widely recognized for its role in facilitating collaborative coding and version control. Here are some key aspects of GitHub:

1. Version Control: GitHub is built on top of the Git version control system. It allows developers to track changes in their codebase, collaborate with others, and manage different versions of their projects. Git enables precise version tracking, branching, and merging, making it easier to work on complex software projects.

2. Code Repositories: GitHub serves as a hosting platform for code repositories. Developers can create repositories to store, manage, and share their source code. These repositories are accessible to team members, contributors, and even the global open-source community.

3. Collaboration: GitHub is designed to foster collaboration. Multiple developers can work on a project simultaneously, making changes to the codebase. GitHub's features include pull requests, which facilitate code reviews and discussion before merging changes into the main codebase. This process promotes code quality and team coordination.

4. Issue Tracking: GitHub includes a built-in issue tracking system. Teams can use it to manage tasks, bugs, feature requests, and other project-related issues. Issues can be assigned, prioritized, and linked to specific code changes, allowing for efficient project management.

5. Projects: GitHub offers project management tools that enable teams to create Kanban boards and track the progress of tasks and features. This helps with project organization and provides a visual way to manage work items.

6. Wiki and Documentation: Developers can create wikis and documentation pages within their repositories, making it easy to share project-related information, guidelines, and user manuals.

7. Continuous Integration (CI): GitHub integrates with various CI/CD (Continuous Integration/Continuous Deployment) services. This allows developers to automate the build, test, and deployment processes, ensuring that code changes are continuously integrated and tested.

8. Community and Open Source: GitHub is a global hub for open-source software development. It hosts millions of open-source projects, making it easy for

contributors from around the world to collaborate and contribute to these projects.

9. Security and Access Control: GitHub provides security features like two-factor authentication and access control mechanisms to protect code repositories. It allows project owners to control who can access, edit, and merge changes into the codebase.

10. Social Features: Developers can follow projects, developers, and organizations on GitHub, making it a social network of sorts for the software development community. This helps keep track of projects, receive notifications, and stay connected with other developers.

In summary, GitHub is a powerful and versatile platform that empowers software developers and teams to manage code, collaborate effectively, and streamline the software development process. It has played a significant role in the open-source community and is widely used in both open-source and private software development projects.

Some remarkable features of GitHub are –
• Commit History can be seen
• Graphs: pulse, contributors, commits, code frequency, members of it. • Pull requests with code review and comments
• Issue Tracking
• Email notifications

Functioning of Git repositories

1. Distributed System and Central Repository:
Distributed systems allow users to collaborate on projects from anywhere in the world. A central repository is a key component in this setup, and it's accessible to remote collaborators using a Version Control System (VCS). Git is one such VCS, known for its popularity. The central repository serves as the hub for project files and version history.

2. Data Redundancy for Reliability:
One challenge with a central server is the risk of data loss or disconnection in case of a failure of the central server. Git addresses this by creating a copy of the entire repository on each snapshot or version that users pull from the central repository. This redundancy ensures that even if the central server crashes, the latest project versions can be recovered from users who have downloaded the most recent snapshots.

3. Simultaneous Collaboration:
Git, as a distributed system, allows multiple users to work on the same project concurrently

without interfering with each other's work. When one user completes their part of the code, they can push these changes back to the central repository. These changes are then updated in the local copies of every other remote user who pulls the latest version of the project.

4. Repository Concept:
Git repositories contain a collection of files representing various versions of a project. The repository holds the complete history and all the changes made to the project's files. Users import these files from the repository to their local machines for further updates and modifications.

5. Version Control System (VCS):
A VCS, such as Git, is responsible for creating and managing different versions of files and storing them in a specific location known as the repository.

6. Cloning and Local Copy:
Cloning in Git is the process of copying the content from an existing Git Repository to the local machine. Once the cloning process is complete, the user has the entire repository on
Name:Chaitanya Kakade
Roll No.:2103076
Batch:C21

their local computer. By default, Git assumes that all work will be performed on the local copy after cloning.

7. Creating and Deleting Repositories:
Git users have the option to create new repositories for their projects or delete existing ones. When deleting a repository, it's as simple as removing the folder containing that repository.

8. Types of Repositories:
Git repositories can be categorized into two types based on their usage on a server: -
Bare Repositories: These repositories are used for sharing changes among different developers. Users are not allowed to modify this repository directly or create new versions based on their modifications.
- Non-bare Repositories: Non-bare repositories are user-friendly and allow users to create new modifications and versions for the repository. During the cloning process, if no specific parameters are provided, Git creates a non-bare repository by default.

Synchronizing with Remote Repositories:

Git allows the users to perform operations on the Repositories by cloning them on the local machine. This will result in the creation of various different copies of the project. These copies are stored on the local machine and hence, the users will not be able to sync their changes with other developers. To overcome this problem, Git allows performing syncing of

these local repositories with the remote repositories.

This synchronization can be done by the use of two commands in the Git. These commands are:

Push: This command is used to push all the commits of the current repository to the tracked remote repository. This command can be used to push your repository to multiple repositories at once. To push all the contents of our local repository that belong to the master branch to the server(Global repository).

Pull: Pull command is used to fetch the commits from a remote repository and stores them in the remote branches. There might be a case when other users perform changes on their copy of repositories and upload them with other remote repositories. But in that case, your copy of the repository will become out of date. Hence, to re-synchronize your copy of the repository with the remote repository, the user has to just use the git pull command to fetch the content of the remote repository.

Git Version Control
Git is software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development.

Its goals include speed, data integrity, and support for distributed, non-linear workflows (thousands of parallel branches running on different systems). Git was created by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development. Since 2005, Junio Hamano has been the core maintainer. As with most other distributed version control systems, and unlike most client–server systems, every Git directory on every computer is a full-fledged repository with complete history and full version-tracking abilities, independent of network access or a central server. Git is free and open-source software distributed under GNU General Public License Version 2.

Some Git Commands:

• Add commands:
$ git add File-name
• Add all files in specific directory to staging area: $ git add –all
$ git add docs/*.txt
• Committing changes:
$ git commit -m "Add existing file"
• Pushing changes:
$ git push -u origin main
• Pulling changes:

$ git pull

```
Last login: Sun Oct 15 12:08:37 on ttys000
You have new mail.
(base) macbook@Chaitanya-Macbook-Pro ~ % mkdir My-Project
(base) macbook@Chaitanya-Macbook-Pro ~ % cd My-Project
(base) macbook@Chaitanya-Macbook-Pro My-Project % git init
Initialized empty Git repository in /Users/macbook/My-Project/.git/
(base) macbook@Chaitanya-Macbook-Pro My-Project % echo "Project" > sample.txt
(base) macbook@Chaitanya-Macbook-Pro My-Project % git add sample.txt
(base) macbook@Chaitanya-Macbook-Pro My-Project % git commit -m "Initial Commit"
[main (root-commit) 0e003e5] Initial Commit
 1 file changed, 1 insertion(+)
 create mode 100644 sample.txt
(base) macbook@Chaitanya-Macbook-Pro My-Project % git remote add origin https://github.com/ChaitanyaK77/New
(base) macbook@Chaitanya-Macbook-Pro My-Project % git push -u origin main
Username for 'https://github.com': ChaitanyaK77
Password for 'https://ChaitanyaK77@github.com':
remote: Support for password authentication was removed on August 13, 2021.
remote: Please see https://docs.github.com/en/get-started/getting-started-with-git/about-remote-repositories#cloning-with-https-urls for information on currently recommended modes
 of authentication.
fatal: Authentication failed for 'https://github.com/ChaitanyaK77/New/'
(base) macbook@Chaitanya-Macbook-Pro My-Project % git push -u origin main
Username for 'https://github.com': ChaitanyaK77
Password for 'https://ChaitanyaK77@github.com':
remote: Support for password authentication was removed on August 13, 2021.
remote: Please see https://docs.github.com/en/get-started/getting-started-with-git/about-remote-repositories#cloning-with-https-urls for information on currently recommended modes
 of authentication.
fatal: Authentication failed for 'https://github.com/ChaitanyaK77/New/'
(base) macbook@Chaitanya-Macbook-Pro My-Project % git push -u origin main
Username for 'https://github.com': ChaitanyaK77/
Password for 'https://ChaitanyaK77/@github.com':
```

EXPERIMENT 12

AIM: To develop Test Cases for the project using White Box Testing(JUnit).

THEORY:

White-box testing is a software testing approach that examines the internal structures, code, and logic of a software application to ensure its correctness and reliability. It is also known as "glass box testing" or "clear box testing" because it involves a deep understanding of the code, allowing testers to design test cases based on knowledge of the software's architecture, data structures, algorithms, and code paths. White-box testing is typically conducted by developers and quality assurance engineers, and it aims to uncover issues such as logic errors, missing functionalities, and security vulnerabilities within the application. It includes techniques like code coverage analysis, path testing, and data flow analysis to assess the code's thoroughness.

JUnit, on the other hand, is a widely used testing framework for Java applications that facilitates the implementation of white-box testing. It provides a robust platform for writing and executing unit tests, a critical aspect of white-box testing. JUnit allows developers to create test cases and suites to verify that individual units of code, like methods and classes, perform as intended. Test cases in JUnit are written as Java methods and utilize annotations and assertions to validate expected outcomes against actual results. The framework simplifies the setup, execution, and reporting of tests, making it an essential tool for ensuring the reliability and correctness of Java applications through comprehensive white-box testing. JUnit also supports test-driven development (TDD) practices, where tests are written before the actual code, helping developers design and validate code incrementally. In summary, JUnit is a crucial component of white-box testing in Java development, providing a structured and systematic approach to testing the internal logic and functionality of software.

Name:Chaitanya Kakade

When working with JUnit for white-box testing, you can follow these steps to create and execute unit tests:

1. Setup and Configuration:

Ensure you have a Java development environment set up with JUnit integrated. You can use build tools like Maven or Gradle to manage dependencies, including JUnit. Make sure your project is configured to use JUnit.

2. Create Test Class:

Start by creating a test class in the same package as the class you want to test. The test class should have the same name as the class under test but with "Test" appended. For example, if you're testing a class named `MyClass`, your test class should be named `MyClassTest`.

3. Import JUnit and Annotate Test Methods:

Import the JUnit classes and annotate your test methods with `@Test`. JUnit provides annotations like `@Before`, `@After`, and `@BeforeClass` for setup and teardown methods. These annotations allow you to perform actions before and after test methods or once for the entire test class.

4. Write Test Methods:

In your test methods, create instances of the class under test, call its methods, and use JUnit's assertion methods like `assertEquals`, `assertTrue`, or `assertFalse` to verify expected outcomes. Ensure that you cover various scenarios, including edge cases and boundary conditions.

5. Execute Tests:

Run the tests using your integrated development environment (IDE)

or build tool. Most IDEs, including Eclipse and VSCode, have built-in

support for running JUnit tests. You can also run tests using the `mvn test` command if you're using Maven.

6. Review Results:

Review the test results. JUnit will provide a summary of passed and failed tests, including any exceptions or failures. Analyze the test results to identify and fix issues in your code.

7. Iterate and Refactor:

If any tests fail, go back to your code, make the necessary changes to address the issues, and re-run the tests. Continue this process until all tests pass.

8. Test Coverage:

Consider using code coverage tools to ensure that your tests cover a sufficient percentage of your code. Aim for high code coverage to verify that your white-box testing is thorough.

9. Continuous Integration:

Integrate your JUnit tests into your continuous integration (CI) pipeline. CI systems like Jenkins and Travis CI can automatically run your tests whenever changes are pushed to the code repository.

10. Documentation and Reporting:

Maintain documentation for your tests, including descriptions of what each test is verifying. Use JUnit's features to generate reports for your tests, and share these reports with your team.

By following these steps and integrating JUnit into your development workflow, you can ensure that your Java code is thoroughly tested from a white-box perspective, helping to identify and address issues early in

the development process.

CODE:

DiseaseClassifier.java inside a root folder named JUNIT

“Path:/Users/macbook/Desktop/JUNIT/DiseaseClassifier.java

”

```
public class DiseaseClassifier {
public static String classifyDisease(String[] symptoms) {
if (contains(symptoms, "Fever") && contains(symptoms, "Cough"))
{ return "Flu";
} else if (contains(symptoms, "Sneezing") && contains(symptoms, "Cough"))
{ return "Cold";
} else if (contains(symptoms, "Sneezing") && contains(symptoms, "Itchy Eyes"))
{ return "Allergies";
} else {
return "Unknown";
}
}

private static boolean contains(String[] arr, String value) {
for (String s : arr) {
if (s.equals(value)) {
return true;
}
}
return false;
}

public static void main(String[] args) {

String[] symptoms = { "Fever", "Cough" };
String result = classifyDisease(symptoms);
System.out.println("The disease is: " + result);
}
}
```

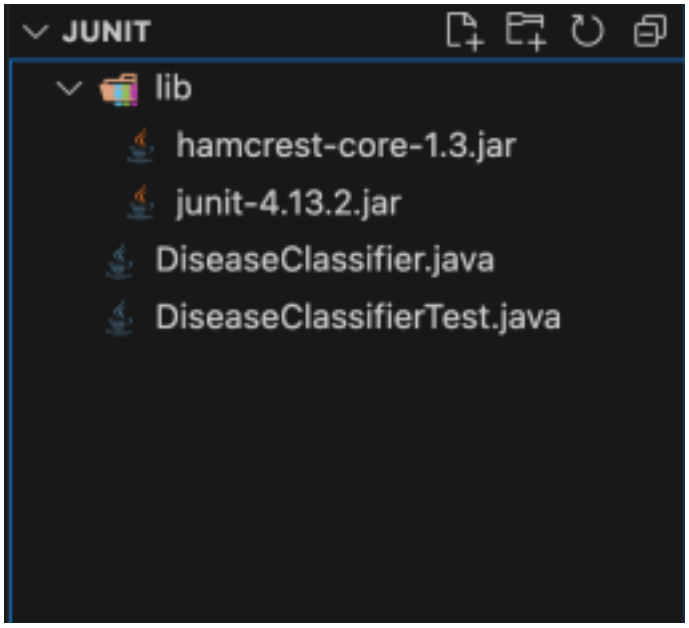
AMAN KARLUPIA
C21
2103084

DiseaseClassifierTest.java inside root folder JUNIT

“Path:/Users/macbook/Desktop/JUNIT/DiseaseClassifierTest.java ” Code:

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class DiseaseClassifierTest {

@Test
public void testFluClassification() {
String[] symptoms = { "Fever", "Cough" };
String result =
DiseaseClassifier.classifyDisease(symptoms);
assertEquals("Flu", result);
}

@Test
public void testColdClassification() {
String[] symptoms = { "Sneezing", "Cough" };
String result =
DiseaseClassifier.classifyDisease(symptoms);
assertEquals("Cold", result);
}

@Test
public void testAllergiesClassification() {
String[] symptoms = { "Sneezing", "Itchy Eyes" };
String result =
DiseaseClassifier.classifyDisease(symptoms);
assertEquals("Allergies", result);
}

@Test
public void testUnknownClassification() {
String[] symptoms = { "Headache", "Fatigue" };
String result =
DiseaseClassifier.classifyDisease(symptoms);
assertEquals("Flu", result);
}
}
```

AMAN KARLUPIA
C21
2103084



Test cases:

(wrong)

```
25      }
26
27      @Test
28      public void testUnknownClassification() {
29          String[] symptoms = { "Headache", "Fatigue" };
30          String result = DiseaseClassifier.classifyDisease(symptoms);
31          assertEquals("Flu", result);    Expected [Flu] but was [Unknown]
32      }
33  }
34
```

(Right)

```
@Test
public void testFluClassification() {
    String[] symptons = { "Fever", "Cough" };
    String result = DiseaseClassifier.classifyDisease(symptons);
    assertEquals("Flu", result);
}

@Test
public void testColdClassification() {
    String[] symptons = { "Sneezing", "Cough" };
    String result = DiseaseClassifier.classifyDisease(symptons);
    assertEquals("Cold", result);
}

@Test
public void testAllergiesClassification() {
    String[] symptons = { "Sneezing", "Itchy Eyes" };
    String result = DiseaseClassifier.classifyDisease(symptons);
    assertEquals("Allergies", result);
}
```

Ehsan 4/10/23 A

Assignment 2

Q. Explain in detail :

a) software Maintenance

software Maintenance is a part of the software development life cycle. Its primary goal is to modify and update software application after delivery to correct errors and to improve performance. software is the model of real world. When the real world changes, the software require alteration wherever possible. software Maintenace is a part of the software. software Maintenance is an inclusive activity that includes error corrections, enhancement of capabilities, deletion of obsolete capabilities, and optimization.

Need for Maintenance:

- correct errors
- change in user requirement with time.
- changing hardware/software requirements.
- To improve system efficiency.
- To optimize the code to run faster.
- To modify the components.
- To reduce any unwanted side effects.

Thus the maintenance is required to ensure that the system continues to satisfy user requirements.

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

Types of software Maintenance:

1) Corrective Maintenance:
Corrective maintenance aims to correct any remaining errors regardless of where they may cause specifications design, coding, testing and documentation etc.

2) Adaptive Maintenance:
It contains modifying the software to match changes in the ever-changing environment.

3) Preventive Maintenance:
It is the process by which we prevent our system from being obsolete. It involves the concept of reengineering and reverse engineering in which an old system with old technology is re-engineered using new technology. This maintenance prevents the system from dying out.

4) Perfective Maintenance:
It defines improving procesing efficiency or performance or restricting the software to enhance changeability. This may contain enhancement of existing systems functionality, improvement in computational efficiency, etc.

2) Re-Engineering:

software re-engineering is a process of software development which is done to improve the maintainability of a software system. Re-engineering is the examination and alteration of a system to reconstitute it in a new form. This process encompasses a combination of sub-processes like reverse engineering, forward engineering, reconstructing etc..

Re-engineering, also known as reverse engineering or software re-engineering, is the process of analyzing, designing and modifying existing software systems to improve their quality performance and maintability. The primary goal of software re-engineering is to improve the quality and maintenability of the software system, while minimizing the risk and costs associated with the redevelopment of the system from scratch.

steps involved in re-engineering are:

1) Planning:
The first step is to plan the reengineering process, which involves identifying the reasons for re-egineering, defining the scope and establishing the goals and objectives of the process.

2) Analysis: The next step is to analyze the existing system, including the code, documentation and other artifacts. This involve identifying the system's strengths and weaknesses as well as any issues that need to be addressed.

3) design:
Based on the analysis, the next step is to design the new or updated software system. This involves identifying the changes that need to be made and developing a plan to implement them.

4) implementation:
The next step is to implement the changes by modifying the existing code, adding new features, and updating the documentation and other artifacts.

5) Testing:
Once the changes have been implemented, the software system needs to be tested to ensure that it meets the new requirements and specifications.

6) Deployment:
The final step is to deploy the re-engineering software system and make it available to end user.

Re-engineering cost factors:
- The quality of the software to be re-engineered.
- The tool support available for re-engineering.
- The extent of the required data conversion.
- The availability of expert staff for re-engineering.

3) Reverse Engineering:

Reverse engineering is a process of recovering the design, requirement specifications, and functions of a product from an analysis of its code. It builds a program database and generates information from this.
The purpose of reverse engineering is to facilitate the maintenance work by improving the understandability of a system and producing the necessary document for a legacy system.

Reverse Engineering goals:

- cope with complexity.
- Recovere lost information.
- Detect side effects.
- synthesise higher abstraction.
- facilitate reuse

steps:

1) collection information:
This step focuses on collecting all possible information (ie., source design documents, etc.) about the software.

2) Examining the information:
The information collected in step 1 studied so as to get familiar with the system.

3) Extracting the structure:
This step concerns identifying program structure in the form of a structure chart where each node corresponds to some routine.

4) Recording the functionality:
During this step processing details of each module of the structure, charts are recorded using structured language like decision table, etc.

5) Recording data flow:
From the information extracted in step 3 and step 4 a set of data flow diagrams is derived to show the flow of data among the processes.

6) Recording control flow:
The high level control structure of the software is recorded.

7) Review extracted design:
The design document extracted is reviewed several times to ensure consistency and correctness. It also ensures that the design represents the program.

8) Generate documentation:
Finally, in this step, the complete documentation including SRS, design document, history, overview, etc is recorded for future use.

Aman karnpia
C21
84

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

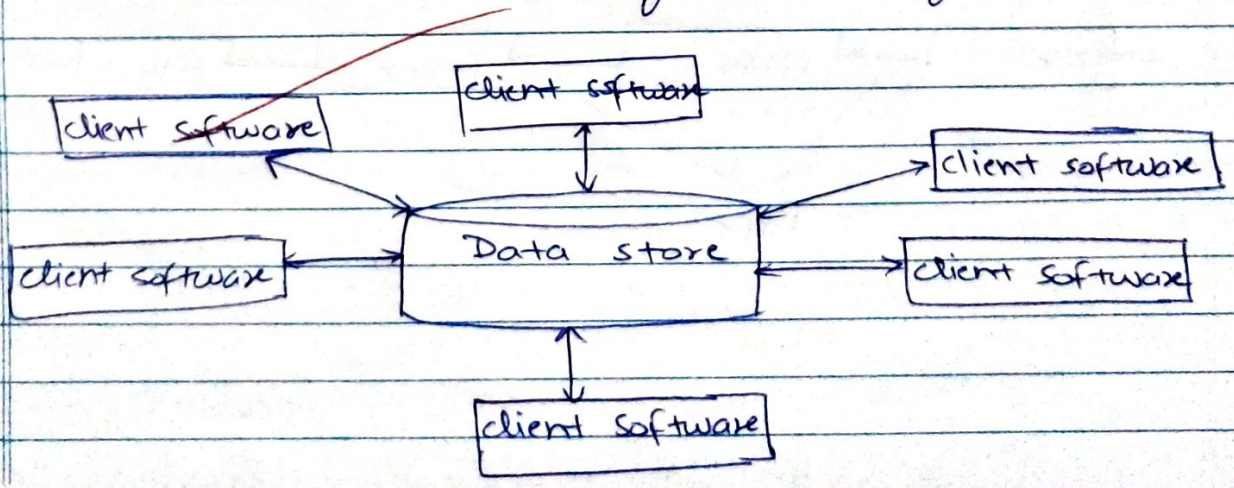
Assignment-1

Q: Architectural Design: Explain in detail each type with examples

The software needs the architectural design to represents the design of software. IEEE defines architectural design as "the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.

▷ Data centered Architecture:

- A data store will reside at the center of this architecture and is accessed frequently by the other components that update, add, delete or modify the data present within the store.
- The figure illustrates a typical data centered style. The client software access a central repository. Variation of this approach are used to transform the repository into a blackboard when data related to client or data of interest for the client change the notifications to client software.
- Data can be passed among clients using blackboard mechanism



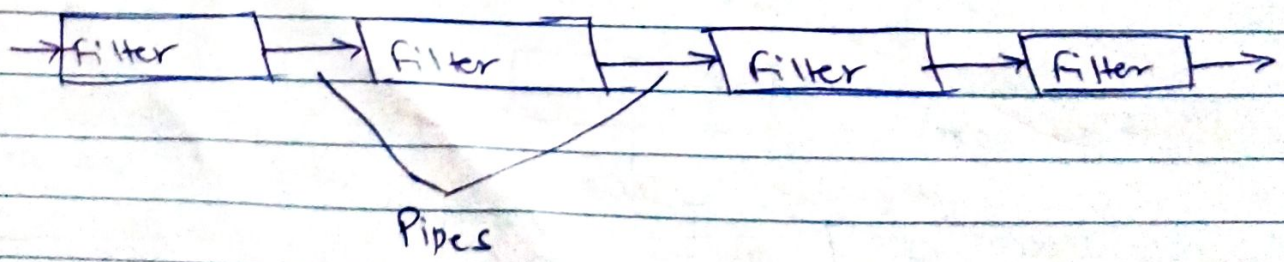
THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

⇒ Data Flow Architecture

- This kind of architecture is used when input data is transformed into output data through a series of computational manipulative components.
- The figure represents pipe-and-filter architecture since it uses both pipe and filter and it has a set of components called filters connected by lines
- Pipes are used to transmitting data from one component to the next.
- If the data flow degenerates into a single line of transforms, then it is termed as batch sequential. This structure accepts the batch of data and then applies a series of sequential components to transform it.

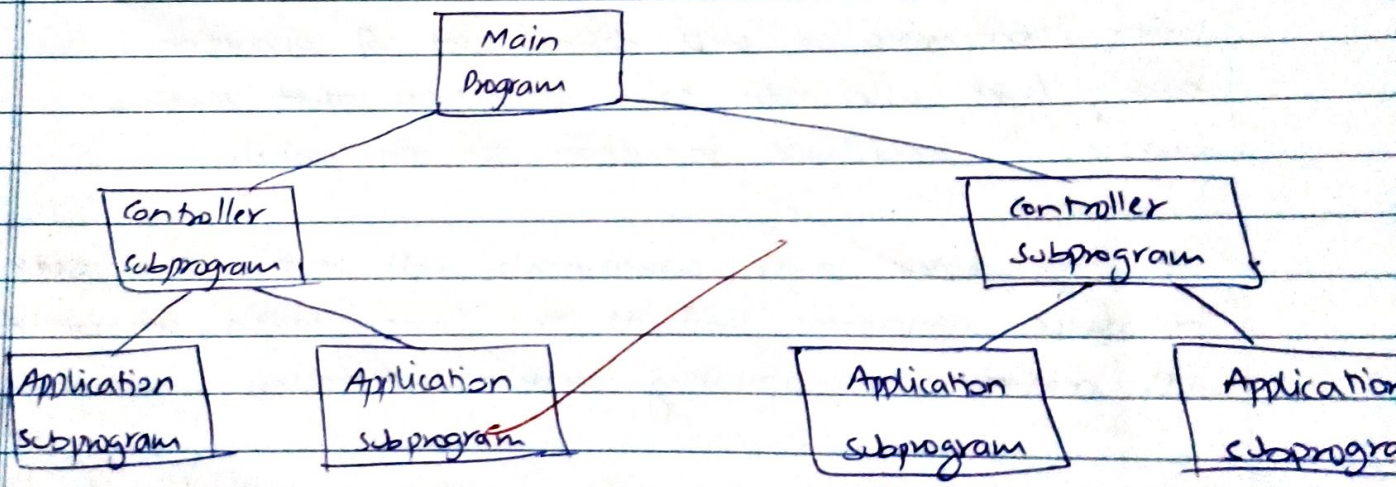
Advantages:

- ⇒ encourages upkeep, repurposing and modification
- with this design, concurrent execution is supported



3) Call and return architecture:
It is used to creat a program that is easy to scale and modify. Many sub-style exists within this category.

- Remote Procedure call architecture:
This components is used to present in a main program or sub program architecture distributed among multiple computers on a network.
- Main Program or subprogram architectures:
The main program structure decomposes into no. of subprograms or function into a control hierarchy. Main program contains no. of subprograms that can invoke other components.



4) object oriented architecture:
The components of a system encapsulate data and the operation that must be applied to manipulate the data. The coordination and communication b/w the components are established via the message passing.

characteristics:
- object protect the system's integrity.
- An object is unaware of the depiction of other items.

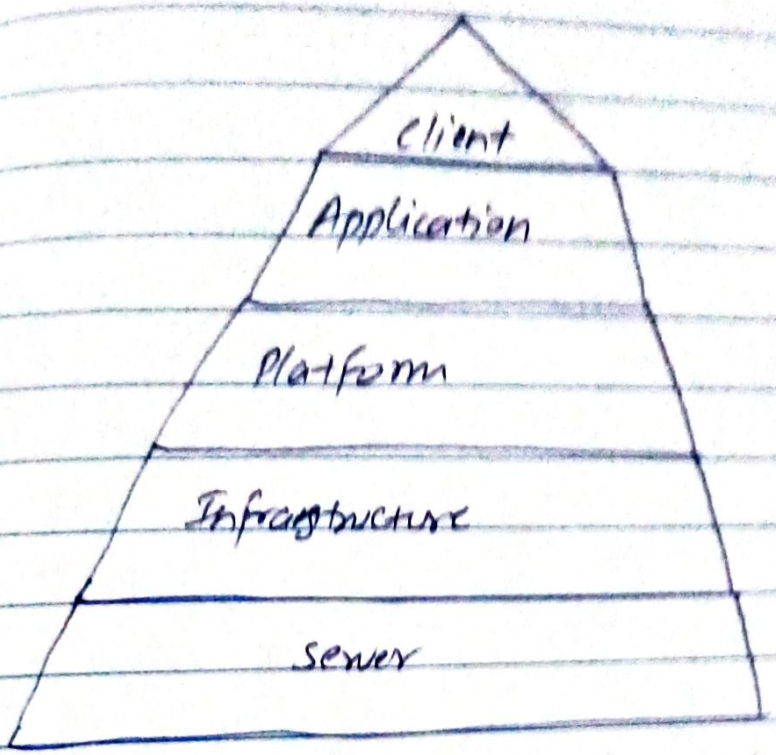
5) layered Architecture:

- A number of diff layers are defined with each layer performing a well-defined set of operations. each layer will do some operations that become closer to machine instruction set progressively.

- At the outer layer, components will receive the user interface operations and at the inner layers, components will perform the operating system interfacing.

- Intermediate layers to utility services and application software functions.

- one common example of this architectural style is OSI-ISO (open systems interconnection - international organisation for standardisation) communication systems.



layered architecture