SOFTWARE ENGINEERING PROJECT

Project Report on Online Courses Recommendation System

SUBMITTED BY

Harsh Arora (22001570006)
Nishant Pratap Singh (22001570018)
Mayank Kumar (22001570022)

B.SC. (HONS.) COMPUTER SCIENCE

Under the Guidance of:
Dr. Vibha Gaur

Department of Computer Science
ACHARYA NARENDRA DEV COLLEGE
University of Delhi,Delhi 110019

ACKNOWLEDGEMENT

We would like to express our deepest gratitude to all those who contributed to the success of this project.

A special note of thanks goes to our mentor, Dr. Vibha Gaur, for providing invaluable guidance, insightful feedback, and continuous support. Your expertise and encouragement were crucial in overcoming challenges and achieving our objectives.

We would also like to extend our appreciation to the department staff, whose assistance and resources were vital in facilitating this project. Your support was indispensable in navigating the logistical and administrative aspects of our work.

Finally, we are grateful to everyone who, directly or indirectly, contributed to this project. Your encouragement and help have been greatly appreciated.

Thank you all for your contributions and support.

Harsh Arora	Nishant Pratap Singh	Mayank Kumar
AE-1218	AE-1233	AE-1232
22001570006	22001570018	22001570022

ACHARYA NARENDRA DEV COLLEGE

(University of Delhi)

CERTIFICATE

This is to certify that Harsh Arora, Nishant Pratap Singh and Mayank Kumar students of B.Sc. (Hons) Computer Science 3rd Year, Acharya Narendra Dev College, University Of Delhi, have completed the Research Project titled 'Online Courses Recommendation System'

HARSH ARORA NISHANT PRATAP SINGH MAYANK KUMAR

Supervisor

Dr. Vibha Gaur

Department of Computer Science

Contents

Chapter 1

PROBLEM STATEMENT

In the rapidly evolving online education landscape, students are often overwhelmed by the sheer number of courses available across various fields. Many struggle to identify which courses are best suited to their current knowledge, future goals, and financial constraints. Just as we rely on recommendation systems in entertainment to find what's best for us, there is a growing need for a similar system in education. However, traditional recommendation systems fall short, as they rely primarily on user behavior and historical data, failing to fully understand the complex factors that determine the best course options for students.

This project aims to use Graph Neural Networks (GNNs) and rule-based mining for course recommendations. While traditional systems focus on users' past activities, GNNs go further by analyzing how courses are interconnected. In this project, courses, learners, and providers are represented as nodes in a graph, with relationships like prerequisites and learning paths as edges. This helps the system understand the broader educational context and provide more informed recommendations.

To improve these recommendations, rule-based mining is used to find patterns and ensure they align with established learning paths. By combining GNNs with rule-based mining, the system offers personalized course suggestions that are relevant and aligned with educational standards.

Chapter 2

PROCESS MODEL

The process model refers to the methodologies and frameworks that guide the planning, execution, and delivery of software projects. A well-defined process model is crucial for ensuring project success, managing resources effectively, and meeting stakeholder expectations. [4]

2.1 Incremental Process Model

The **Incremental Process Model** is a software development methodology that emphasizes building a system in small, manageable increments or phases. Each increment adds specific features or functionalities to the existing system, allowing for partial deployment of the software even before the complete product is finalized. This approach fosters iterative development, where each increment undergoes testing and refinement based on user feedback, ensuring that the final product aligns closely with user needs and expectations.

In the context of an Online Courses Recommendation System, the Incremental Process Model allows for the gradual development and enhancement of the recommendation engine, integrating new features and capabilities based on continuous user interaction and evolving requirements.

2.1.1 Motivation for Employing the Incremental Process Model

a. Manageable Development Phases

i. **Focus on Individual Components**: The model allows the development team to concentrate on specific features, reducing the complexity associated with building the entire system at once.

ii. **Controlled Progression**: Each increment is developed, tested, and refined independently, ensuring that progress is manageable and well-structured.

b. Early Delivery of Functionalities

i. **Rapid Initial Release**: The system can be deployed with core functionalities early in the development cycle, enabling users to start benefiting from its features while subsequent increments are developed.

ii. **Continuous Value Addition**: Each increment introduces new features and improvements, consistently enhancing the overall value of the recommendation system for users.

c. Continuous User Feedback

i. **User-Centric Development**: User feedback is collected after each increment, allowing the team to adjust features and functionalities based on actual user experiences and needs.

ii. **Responsive Refinement**: Insights gained from user interactions enable the development team to refine and enhance the system, ensuring it remains relevant and effective.

d. Risk Mitigation

i. **Early Issue Detection**: Potential problems can be identified and addressed early in the development process, reducing the risk of significant issues arising later.

ii. **Flexibility in Change Management**: Changes based on user feedback can be integrated smoothly into future increments, ensuring the project stays aligned with user expectations.

e. Adaptability to Changes

i. **Evolving Educational Needs**: The dynamic nature of online education requires the system to adapt quickly to changing user preferences and course offerings. The incremental approach facilitates this adaptability.

ii. **Feature Prioritization**: Development priorities can be reassessed after each increment, allowing the team to focus on the most valuable features in response to changing market trends.

f. Cost-Effective Development

  i. **Efficient Resource Allocation**: The team can allocate resources more effectively by focusing on high-priority features during each increment, optimizing development efforts.

  ii. **Minimized Over-Engineering**: The risk of over-engineering is reduced as each feature is evaluated for its necessity based on user feedback and interactions.

g. Incremental Testing and Quality Assurance

  i. **Regular Testing Opportunities**: Each increment undergoes thorough testing, ensuring that the system maintains high quality and reliability throughout development.

  ii. **Building on Solid Foundations**: As increments are built on the previous ones, the overall stability and performance of the recommendation system improve with each release.

h. Enhanced Stakeholder Engagement

  i. **Frequent Progress Updates**: Stakeholders receive regular updates on development progress, fostering transparency and maintaining their interest in the project.

  ii. **Encouraging Collaboration**: The model promotes collaboration among the development team, stakeholders, and users, aligning efforts toward a shared vision for the final product.

2.2 Requirement Analysis and Modeling

Requirement Analysis and Modeling is a critical phase in the software development process that focuses on understanding and documenting the needs and expectations of stakeholders for a particular system. This process involves identifying both functional requirements—specific features and functionalities the system must provide—and non-functional requirements, which describe the system's performance, usability, security, and other quality attributes.

Requirement analysis aims to bridge the gap between stakeholder needs and technical specifications, ensuring that the final product aligns with user expectations and business objectives. Modeling, on the other hand, involves creating visual representations, such as use case diagrams, user stories, and data flow diagrams, which help in conceptualizing how the system will function and interact with its users. This structured approach facilitates clearer communication among stakeholders and developers, reduces the risk of misunderstandings, and provides a foundation for subsequent design and implementation activities. [8]

2.3 DFD

Data Flow Diagram (DFD) represents the flow of data within information systems. Data Flow Diagrams (DFD) provide a graphical representation of the data flow of a system that can be understood by both technical and non-technical users. The models enable software engineers, customers, and users to work together effectively during the analysis and specification of requirements.

2.3.1 Context Level DFD

Context Level is the highest-level Data Flow Diagram (DFD), which provides an overview of the entire system. It shows the major processes, data flows, and data stores in the system, without providing any details about the internal workings of these processes. The DFD regarding context level is shown in figure 2.1

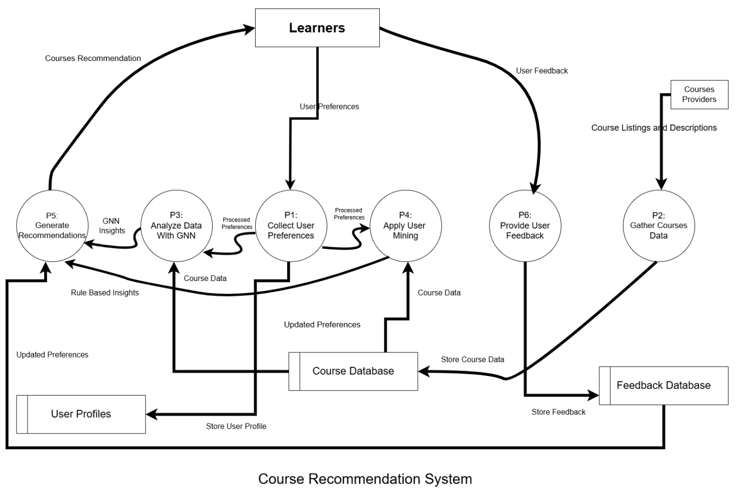

Figure 2.1: Context Level Data Flow Diagram

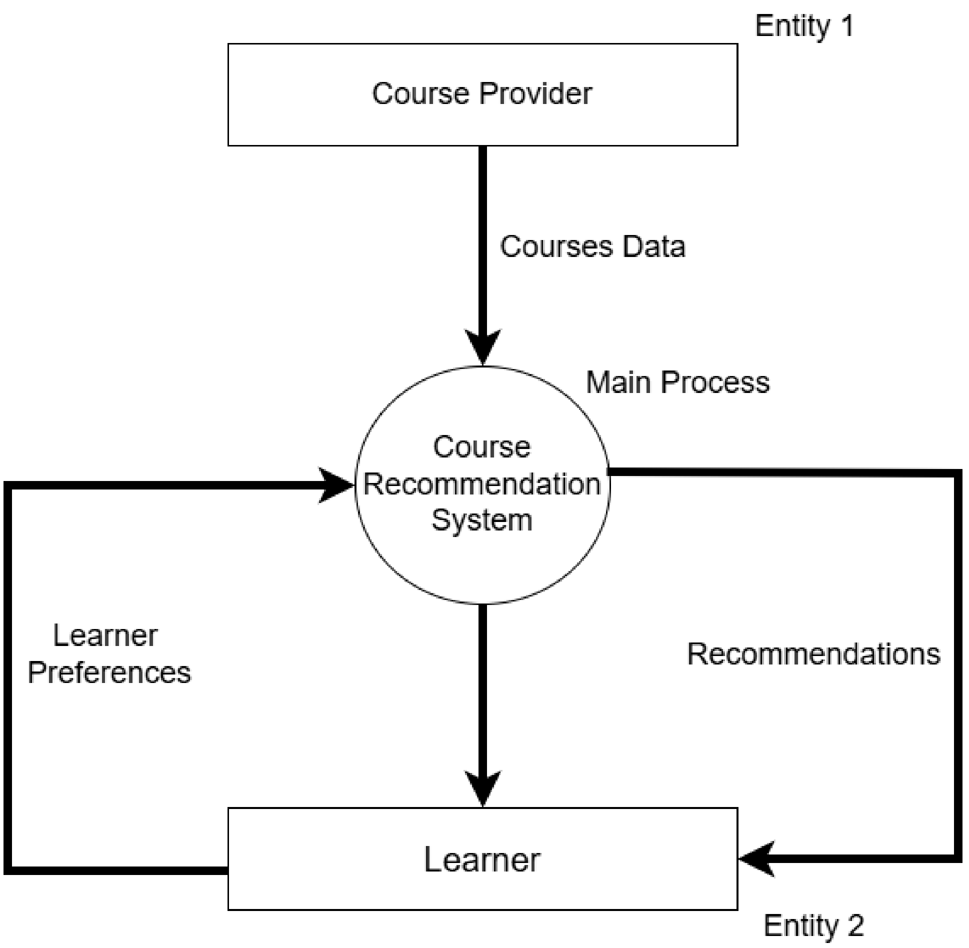

Figure 2.2: Level 1 Data Flow Diagram

2.3.2 Level 1 DFD

1-Level provides a more detailed view of the system by breaking down the major processes identified in the level 0 Data Flow Diagram (DFD) into sub-processes. Each sub-process is depicted as a separate process on the level 1 Data Flow Diagram (DFD). The data flows and data stores associated with each sub-process are also shown in figure 2.2.

2.4 Data Dictionary

A Data Dictionary is a collection of metadata that describes the structure, content, and meaning of data elements within a database, information system, or research project.Our DD is shown in Table 2.1 and Table 2.2

Table 2.1: Data Dictionary for the Online Courses Recommendation System (Part 1)

Attribute Name	**Components**
Learner ID	Positive integer
Learner Profile	[Learner ID + Course Enrollment + User Preferences + Feedback + GNN Insights + Rule-Based Insights]*
First Name	A–Z + a–z
Last Name	A–Z + a–z
Password	A–Z + a–z + @ + $ + % + & + digit + digit + digit + digit
Credentials File	[User ID + User Name + Email + Password Hash]*
User ID	Digit + Digit
User Name	A–Z + a–z
Email	A–Z + a–z + digit + digit + digit + digit + @gmail.com
User Data	[Student Name + Subject/Course Preferences]*
Subject Preferences	A–Z + a–z
Course Preferences	A–Z + a–z
Content Title	[Course Name]*
Course Name	A–Z + a–z
Content ID	A–Z + a–z + digit + digit + digit
Login Request	User Email + Password
Login Response	success — failure
API Call	API Key + Content Name
API Key	A–Z + a–z + digit + digit + digit + digit
API Result	[Learning Contents]

Table 2.2: Data Dictionary for the Online Courses Recommendation System (Part 2)

Attribute Name	Components
Course Listings	[Course ID + Course Provider ID + Listing Date]*
Course Recommender System	[Learner ID + Course ID + User Preferences + GNN Insights + Rule-Based Insights]*
User Preferences	[Learner ID + Course ID + Preprocessed Preferences]*
Preprocessed Preferences	[Learner ID + Transformed Preferences]*
Feedback	[Learner ID + Course ID + Feedback Text + Feedback Date]*
GNN Insights	[Learner ID + Course ID + Insight Value]*
Rule-Based Insights	[Learner ID + Course ID + Rule-Based Score]*

2.5 Use Case Approach

A use case is a description of how a user interacts with a system to achieve a specific goal. It outlines the steps involved in a particular interaction, the roles of different users (or actors), and the expected outcomes. [6]

2.5.1 Use Cases

Use Case 1: User Registration and Authentication

Actors:

- Learner
- System

Primary Actor: Learner

Preconditions:

- The learner must have access to the internet and a valid email address.

Main Success Scenario:

(i) The learner accesses the registration page.

(ii) The learner provides required information (e.g., email, password, name).

(iii) The system validates the provided details and creates a user profile.

(iv) The learner logs into the system using their credentials.

Exception Scenario:

(i) The email provided by the learner is already registered.

(ii) The password entered does not meet security requirements.

Use Case 2: Collect User Preferences

Actors:

- Learner
- System

Primary Actor: Learner

Preconditions:

- Learner has successfully created an account.

Main Success Scenario:

(i) The learner accesses the recommendation system.

(ii) The system prompts the learner to input their preferences (e.g., subject interests, learning goals, preferred course difficulty, etc.).

(iii) The learner enters their preferences into the system.

(iv) The system preprocesses the input preferences (e.g., normalizes data, checks for consistency).

(v) Preprocessed preferences are stored in the system's database for further analysis and use in generating personalized course recommendations.

Extension Scenario:

(i) The system detects that some required preference fields are incomplete.

(ii) The system prompts the learner to fill in all required fields before proceeding.

Use Case 3: Course Recommendation

Actors:

- Learner
- System

Primary Actor: System

Preconditions:

- The learner must be logged into the system with a completed profile or have interacted with the system to generate data for recommendations.
- The recommendation engine must be operational, with access to the learner's profile data, interaction history, and course database.

Main Success Scenario:

(i) The learner requests course recommendations.

(ii) The system analyzes the learner's profile data, preferences, and course interaction history.

(iii) The system generates and displays personalized course recommendations.

Exception Scenario:

(i) The learner's profile is incomplete or lacks sufficient data for personalized recommendations.

(ii) The recommendation engine fails to retrieve or process data.

2.6 Sequence Diagrams

Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration. Sequence Diagrams are time focus and they show the order of the interaction visually by using the vertical axis of the diagram to represent time what messages are sent and when.
[9]

Chapter 3

SOFTWARE REQUIREMENT SPECIFICATION

The Software Requirement Specification (SRS) is a comprehensive document that outlines the functional and non-functional requirements of a software system. It serves as a blueprint for both developers and stakeholders, detailing what the system is expected to achieve and the criteria for its success.

3.1 Introduction

The Introduction section provides an overview of the SRS document, including the purpose, scope, and definitions related to the system under development. It ensures all stakeholders have a clear understanding of the project goals and requirements. This section covers the following subsections:

3.1.1 Purpose

This project aims to deliver a lightweight and targeted Course Recommendation System focused on providing personalized recommendations for learners based on their preferences and past interactions. The system will allow users to receive suggestions from a predefined, static list of courses from the course database. Additionally, it will provide mechanisms for user feedback and limited profile management to optimize recommendations over time.

3.1.2 Scope

This project involves building a recommendation system that provides users with relevant courses based on their dynamic preferences. The key features include user authentication, input of learning preferences, course recommendations, and feedback collection. The **Graph Neural Network** will serve as the backbone for generating recommendations. The system will be deployed on a mid-tier server and support both Android and iOS devices.

Included in Scope:

(i) Course recommendations based on dynamic user preferences.

(ii) A fully operational mobile app for Android and iOS.

(iii) User authentication and profile management.

(iv) User feedback collection to improve recommendations.

(v) Backend GNN-based recommendation system.

Excluded from Scope:

i Real-time course updates from external providers.

ii Career path suggestions.

iii Integration of dynamic recommender systems or live feedback processing.

3.1.3 Definitions, Acronyms, and Abbreviations

i **GNN**: Graph Neural Network

ii **GDPR**: General Data Protection Regulation

iii **FERPA**: Family Educational Rights and Privacy Act

iv **PostgreSQL**: Open-source relational database management system

3.1.4 Overview

This document covers all the detailed functional and non-functional requirements, system architecture, design constraints, and interface requirements for the project. The development of the system will follow an agile approach, ensuring that it is completed within a two-month timeline.

3.2 General Description

The system will provide course recommendations to users by leveraging a GNN model that analyzes relationships between courses and user preferences. The mobile app will allow users to input their learning preferences, view recommended courses, and provide feedback. The system is a standalone product but must comply with privacy regulations.

3.2.1 Product Functions

i **User Authentication**: Users will be able to log in to the system using an email and password. They will be able to recover passwords if forgotten.

ii **Preference Input**: Users can select or input preferences such as domain, course level, and course duration to receive personalized recommendations.

iii **Course Recommendations**: The system will display a list of recommended courses based on user preferences and learning history.

iv **Feedback Collection**: Users can rate and provide feedback on completed courses to improve future recommendations.

3.2.2 User Characteristics

i **Students**: Individuals looking to enhance their skills through online courses.

ii **Professionals**: Users aiming to expand knowledge in their field or explore new areas of interest.

3.2.3 Constraints

i The system will only support Android and iOS platforms.

ii The backend GNN computation will be hosted on a mid-tier server, which may impose limits on the number of simultaneous users.

3.3 Specific Requirements

This section describes the core functionality of the system, focusing on how the input from users translates into the system's output.

User Authentication

Users must be able to log in with their email and password credentials.

i **Input**: Email (up to 50 characters), Password (up to 15 characters).

ii **Output**: Successful login or error message for incorrect credentials.

iii **Source**: User-provided during the sign-up process.

Course Preferences Input

Users can enter preferences such as their organization, course domain, and course level to personalize their recommendations.

i **Input**: Organization (dropdown), Domain (dropdown), Course Name (input), Rating (dropdown), Level (dropdown), Duration (input).

ii **Output**: A filtered list of courses based on preferences.

iii **Source**: User input.

Course Recommendations

The system will provide personalized recommendations based on user preferences and learning history, powered by the GNN on the backend.

i **Input**: Dynamic user preferences and course database.

ii **Output**: Recommended courses with attributes such as Course Title, Provider, Level, Rating, Domain, Duration.

iii **Source**: Data derived from the GNN model.

Feedback Collection

Users will be able to rate and provide feedback on the courses they complete, which will be stored and used to refine future recommendations.

i **Input**: Course rating (dropdown).

ii **Output**: Confirmation of submitted feedback.

iii **Source**: User input.

3.3.1 Performance Requirements

Static Performance Requirements

i **Terminal/Device Support**: The system must support up to **200 simultaneous users** across both desktop and mobile devices.

ii **Simultaneous Users**: The server must be able to handle **at least 50 concurrent users** interacting with the course recommendation engine without noticeable performance degradation.

Dynamic Performance Requirements

i **Response Time**: The system must provide a course recommendation within **3 seconds** of a user submitting preferences.

ii **Latency**: Any actions taken by the user on the app (login, submitting preferences, etc.) should have a response time of less than **2 seconds**.

3.3.2 Design Constraints

Standard Compliance

i The system must comply with **GDPR** and **FERPA** regulations to ensure the security and privacy of user data.

ii Reports generated for user recommendations and feedback will follow **standard educational reporting formats**.

Hardware Limitations

i The backend system will run on a **mid-tier server** with **16 GB of RAM** and a **mid-tier CPU**. No GPU acceleration is required, as the GNN computations have been optimized for CPU use.

ii The mobile application will function on standard hardware configurations for **Android** and **iOS** devices without requiring high-performance hardware.

Reliability and Fault Tolerance

i **Uptime**: The system must maintain an uptime of **99.5%**, ensuring availability for users.

ii **Fault Tolerance**: In the event of a server crash, the system will back up data every **24 hours** and provide session recovery options when back online.

3.3.3 External Interface Requirements

Hardware Interface

i The system will interface with **desktop** and **mobile devices**. Each terminal must have access to the internet, and for mobile devices, the system must support both **Android** and **iOS**.

ii The server must support mid-tier processing capabilities with **16 GB of RAM**.

Software Interface

i **Backend**: The system backend will be built using **Django**, managing data and course recommendations through a **PostgreSQL** database.

ii **Frontend**: The mobile application frontend will be developed using **Flutter**, ensuring compatibility across Android and iOS platforms.

User Interface

i **Login Screen**: Users will enter their email and password to authenticate.

- Fields: Email, Password

ii **Course Preferences Input**: Users can enter preferences such as organization, domain, and course level.

Communications Interface

The system will communicate over a secure **HTTPS** protocol to ensure the security and privacy of user data during interactions.

3.4 Other Non-Functional Requirements

This section describes the other functionality of the system, focusing on how system will function after final deveopment.

3.4.1 Security

i The system must provide **user authentication** using email and password credentials.

ii The system must use **encryption** for sensitive data such as passwords and user preferences.

3.4.2 Data Privacy

i The system must comply with **GDPR** and **FERPA** regulations for data privacy and protection.

ii All personal data, including course preferences, will be anonymized in aggregated reporting.

3.4.3 Maintainability

i The codebase will be modular, allowing for easy updates to the recommendation algorithm and user interface.

ii Documentation will be maintained for each module to ensure that future developers can easily extend the system.

3.4.4 Scalability

The system should be able to scale in response to an increase in the number of users and courses without major changes to the backend infrastructure.

3.4.5 Usability

The mobile app must be user-friendly, with an intuitive design to ensure seamless interaction for users of all technical backgrounds.

Chapter 4

ESTIMATIONS

The Estimations provides a detailed breakdown of the time, resources, and cost required to complete the project successfully. Accurate estimations are crucial for effective project planning, ensuring that tasks are completed on time and within budget. Estimations typically cover various aspects such as effort, duration, financial costs, and resources, all of which help in managing the project's scope and expectations.

4.1 Function Points

Function Points (FP) are a standardized unit of measurement that quantify the functionality delivered by a software system to its users. They are widely used in software engineering for estimating project size, determining development effort, and assessing productivity. The FP method measures various aspects of the software, including inputs, outputs, user interactions, files, and interfaces, translating them into a numeric value that reflects the system's complexity and functionality.

4.1.1 External Inputs

i User registration and login (authentication module).

ii Course preferences input (dynamic user preferences).

iii Course feedback submission.

4.1.2 External Outputs

i Recommended course list.

ii User feedback confirmation.

iii Profile management and recommendation history.

4.1.3 External Inquiries

i Search functionality for courses based on user preferences.

ii Inquiries about course details (e.g., syllabus, duration, instructor).

iii User queries regarding account status and course recommendations.

4.1.4 Internal Logical Files (ILFs)

i User profiles (authentication data, preferences).

ii Course dataset.

4.1.5 External Interface Files (EIFs)

i External course APIs (if used in future scope).

ii Third-party authentication services (e.g., OAuth for login).

4.1.6 Complexity Weights

i EIs: Medium complexity.

ii EOs: Medium complexity.

iii EQs: Low complexity.

iv ILFs: Low complexity.

v EIFs: Low complexity.

4.2 Information Domain Values Estimation

In this section, we present the estimation of Information Domain Values for the Online Course Recommendation System. These values are based on the complexity factors that influence the system's design and functionality. The following table summarizes the complexity factors and their associated values:

Complexity Factor	**Value**
Backup and recovery	2
Data communications	2
Distributed processing	2
Performance critical	3
Existing operating environment	2
Online data entry	3
Input transactions over multiple screens	3
Master files updated online	3
Information domain values complex	3
Internal processing complex	5
Code designed for reuse	3
Conversion/installation in design	2
Multiple installations	2
Application designed for change	4
Total Count	**38**

Table 4.1: Complexity Factors and Values

The total count of complexity factors amounts to 45. Each value reflects the estimated complexity of different aspects of the system, ranging from backup and recovery needs to user interactions.

4.2.1 Total Unadjusted Function Points

The total unadjusted function points (UFP) for the online course recommendation system are calculated by multiplying the estimated count of each function type by their respective complexity weights. This calculation serves as an important metric to gauge the system's functional size and helps in subsequent effort estimation. The following table summarizes the function point calculations:

Table 4.2: Function Point Calculations for the Online Course Recommendation System

Category	**Estimated Count**	**Weight Factor**	**Calculated Value**
Inputs	3	7	$3 \times 7 = 21$
Outputs	3	5	$3 \times 5 = 15$
Inquiries	2	4	$2 \times 4 = 8$
Internal Logical Files (ILFs)	2	3	$2 \times 3 = 6$
External Interface Files (EIFs)	4	3	$4 \times 3 = 12$
Total			$21 + 15 + 8 + 6 + 12 = 62$

From the calculations, we find that the total unadjusted function points (UFP) can be computed as follows:

$$\text{Total UFP} = 21 + 15 + 8 + 6 + 12 = 62$$

4.2.2 Complexity Adjustment Factor (CAF)

Next, we determine the complexity adjustment factor (CAF), which accounts for various project characteristics that may influence the overall complexity of the system. The CAF is calculated using the following formula:

$$\text{CAF} = 0.65 + (0.01 \times 38) = 0.65 + 0.38 = 1.03$$

This factor incorporates considerations such as the performance requirements, existing operating environment, and other relevant factors impacting the system's complexity.

4.2.3 Total Adjusted Function Points (FP)

Finally, we compute the total adjusted function points (FP) by multiplying the total unadjusted function points by the complexity adjustment factor:

$$\text{FP} = \text{Total UFP} \times \text{CAF}$$

Substituting the values, we find:

$$\text{FP} = 62 \times 1.03 = 63.86$$

Therefore, the estimated function points for the Online Course Recommendation System are:

$$\text{Estimated FP} = 63.86$$

Chapter 5

SCHEDULING

The Scheduling chapter outlines the project timeline, breaking down tasks into manageable phases with specific deadlines and milestones. Proper scheduling is essential for tracking progress, ensuring timely completion of tasks, and maintaining alignment with project goals. This section will also highlight dependencies between tasks and allocate sufficient time for testing, reviews, and unforeseen challenges. [7]

5.1 Key Phases and Tasks

The development of the Online Courses Recommendation System is organized into several key phases, each comprising specific tasks that are crucial for the successful completion of the project. The timeline for these phases is illustrated in the accompanying Gantt chart.

5.1.1 Project Planning and Initialization (Week 1)

In the first week, the project will focus on planning and initializing essential activities. This phase includes:

I. **Meet with Team:** Assemble the project team to discuss objectives and roles.

II. **Identify Project Needs and Constraints:** Define what the project aims to achieve while acknowledging limitations.

III. **Establish Problem Statement:** Clearly articulate the problem the recommendation system intends to solve.

5.1.2 Requirement Analysis (Weeks 1-2)

The requirement analysis phase spans the first two weeks, focusing on gathering and finalizing the system requirements. Key activities include:

I. **Define System Components:** Identify the core components required for the recommendation system.

II. **Review Design with Team:** Collaborate with the team to ensure everyone understands and agrees on the proposed design.

5.1.3 System Design (Weeks 2-3)

During weeks two and three, the focus will shift to system design, which involves:

I. **Design GNN Architecture:** Outline the architecture of the Graph Neural Network to effectively analyze relationships in the data.

II. **Design UI/UX:** Create user interface and experience designs that are intuitive and user-friendly.

5.1.4 Backend Development (Weeks 3-5)

The backend development phase will take place from weeks three to five and will encompass:

I. **Research GNN and Association Mining:** Investigate the relevant literature and techniques for implementing GNN and association mining algorithms.

II. **Implement and Train GNN Model:** Develop the GNN model, ensuring it is trained effectively to provide accurate recommendations.

III. **Integrate ML Backend with Python:** Ensure seamless communication between the machine learning backend.

5.1.5 Frontend Development (Weeks 4-6)

In parallel with backend development, frontend development will occur from weeks four to six, focusing on:

I. **Implement Web APP with ReactJS:** Build the Web application using ReactJS, incorporating all required functionalities.

II. **Define App Functionalities:** Specify and document the functionalities that the Web application will provide to users.

5.1.6 Integration and Testing (Weeks 6-7)

From weeks six to seven, the project will enter the integration and testing phase, which includes:

I. **Conduct System Testing:** Perform comprehensive testing of the entire system to ensure all components work together as intended.

II. **Test App Features:** Verify that all features of the web app.

III. **Perform Unit Testing:** Test individual components of the system .

5.1.7 Finalization and Deployment (Week 8)

In the eighth week, the system will be prepared for finalization and deployment, involving:

I. **Final App Build and Deployment:** Compile the final version of the application and deploy it on the designated platforms.

II. **Prepare Deployment Environment:** Ensure the deployment environment is ready and properly configured for the application launch.

III. **Write Technical Documentation:** Document technical details about the system architecture, installation, and maintenance procedures.

5.1.8 Presentation and Wrap-Up (Weeks 8-9)

Following deployment, the project will transition into the presentation and wrap-up phase, which will occur in weeks eight and nine. Activities in this phase will include:

I. **Final Project Presentation:** Prepare and deliver a presentation that showcases the project's objectives, processes, and outcomes.

II. **Scope Model Input Functions:** Finalize how user inputs will be handled by the system to ensure optimal performance and usability.

III. **Finalize System Output Functions:** Confirm the output that the system will provide based on user inputs and GNN analysis.

5.2 Timeline Overview

The timeline overview for the project phases is as follows:

I. **Week 1**: Project planning and initialization.

II. **Weeks 1-2**: Requirement analysis.

III. **Weeks 2-4**: System design.

IV. **Weeks 3-5**: Frontend development.

V. **Weeks 4-6**: Backend development.

VI. **Weeks 6-7**: Integration and testing.

VII. **Week 7-8**: Finalization and deployment.

VIII. **Weeks 8**: Documentation

5.3 Gantt Chart for Online Course Recommendation System

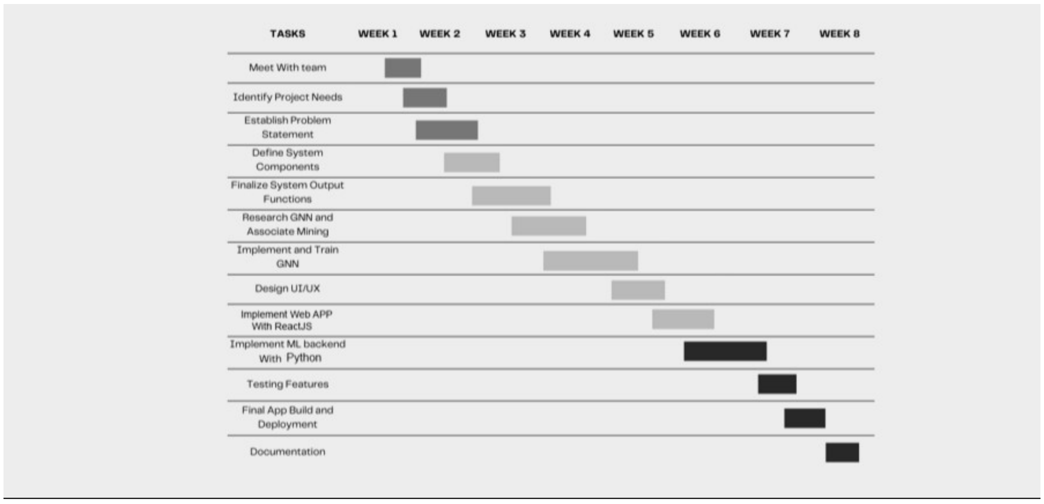

Figure 5.1: Timeline of the project

Chapter 6

Risk Management

Risk management is a crucial aspect of any project, as it helps in identifying, assessing, and mitigating potential threats that could hinder the success of the project. Effective risk management ensures that the project remains on track, within budget, and achieves its goals despite uncertainties or challenges that may arise during its lifecycle. [1]

6.1 Risk Table

The risk table is a vital tool for managing potential challenges that could arise during the project. Each column in the table provides critical information about specific risks and how they will be handled. Below is an explanation of each column:

- **Risks**: This lists the potential risks that could affect the project, identifying specific scenarios or events that may arise and negatively impact progress, quality, or timelines.
- **Category**: The category refers to the type of risk, which can be related to various aspects such as technical, financial, scheduling, or security risks. It helps in classifying risks for better management.
- **Probability**: This denotes the likelihood of the risk occurring. The probability is usually categorized as Low, Medium, or High, providing insight into how likely it is that the risk will manifest during the project lifecycle.
- **Impact**: The impact refers to the potential consequences of the risk on the project, also rated as Low, Medium, or High. It reflects how significantly the project will be affected if the risk occurs.

- **RMMM**: RMMM stands for Risk Mitigation, Monitoring, and Management. This outlines the strategies for mitigating the risk, monitoring it over time, and managing its occurrence. This helps ensure that there are concrete actions to minimize the risk's impact and that it is tracked throughout the project.

Risks	**Category**	**Probability**	**Impact**	**RMMM (Risk Mitigation, Monitoring, and Management)**
Delays in project schedule due to resource constraints	Scheduling	Medium	High	Allocate additional resources and adjust the project timeline to accommodate the delay. Regularly monitor resource availability.
Technical challenges in integrating new technology	Technical	High	Medium	Provide training to the team and allocate extra time for testing and debugging. Monitor integration issues closely.
Changes in customer requirements mid-project	Scope	Low	High	Maintain regular communication with stakeholders. Implement a formal change control process to assess impact before incorporating changes.
Budget overruns due to unforeseen expenses	Financial	Medium	Medium	Establish a contingency fund and regularly review the budget. Adjust resource allocation and scope as necessary.
Data security breach during development	Security	Low	High	Implement strict security protocols such as encryption and access controls. Conduct regular audits and vulnerability assessments.

Table 6.1: Risk Table Example

Chapter 7

DESIGN

The design phase is a critical step in the development of any system. It involves the detailed planning and structuring of components to ensure that the system functions efficiently and meets the intended requirements. In this phase, the architecture of the system is outlined, specifying how different modules will interact with one another and how data will flow between them. [5]

7.1 Structured Chart

A structural chart is a graphical representation of a system's architecture and the relationships between its components or modules. It breaks down the system into smaller parts, showing how the parts interact with each other. Structural charts emphasize the hierarchical structure of the system, providing a top-down view that describes:

How modules or components are related. The flow of data between different modules. The invocation of submodules by higher-level modules.

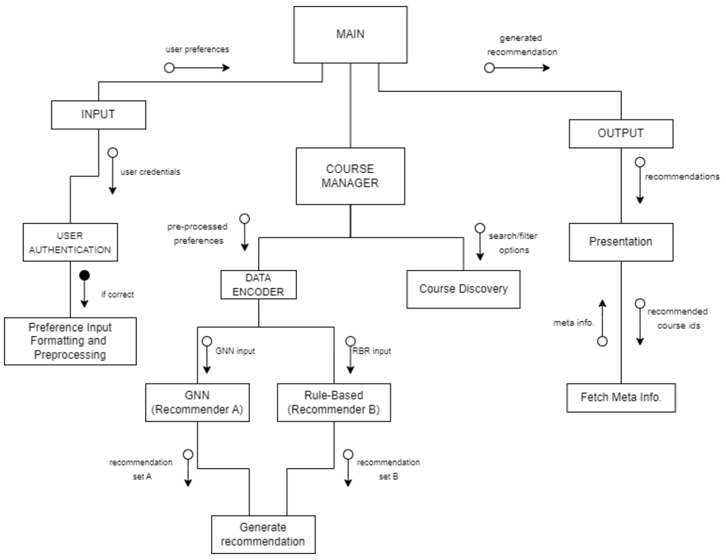

Figure 7.1: Structural chart of the Online Course Recommendation system

7.2 Pseudo Code

Pseudocode is a method used to describe an algorithm or process using plain English, rather than writing actual code. It's essentially a rough draft or blueprint for a program that helps plan and design algorithms and prototypes.

With reference to figure 7.1 , the proposed pseudocode is as follows:

7.2.1 Input Module

```
1. Show the login page.
2. Ask the user to enter the login credentials.
3. If the credentials are correct:
    a. Show a Preference page to the user.
    b. Get the user's prefernces.
4. Send the user's preferences to the main module.
```

7.2.2 Main Module

```
1. If preferences are provided:
```

   a. Receives the user's preferences.
   b. Send the user's preferences to the Data Encoder:
      i. Encode the raw preferences into the format required for the recommendation process.
   c. Pass the encoded data to the recommendation process:
      i. Send the encoded data to the GNN model and generate GNN-based recommendations.
      ii. Send the encoded data to the Rule-based model and generate rule-based recommendations.
   d. Send both the GNN and Rule-based recommendations to the output module.

2. If preferences are not provided (course discovery mode):
   a. Discover courses based on general courses.

7.2.3 Output Module

1. Collect the recommendations from both the GNN model and the rule-based model.
2. Provide a user-friendly interface that displays the recommended courses, possibly with options like, subject, domain, links, level, etc.
3. Allow the user to select a course for more details.
4. Capture user feedback on the recommendations to refine future recommendations.

Chapter 8

CODING

In this chapter, we will delve into the essential role of coding in developing robust applications and systems. Coding is the backbone of software development, enabling us to transform ideas into functional programs. Through various code snippets, we will explore the practical implementations of algorithms and techniques that are critical for our project.

8.1 Code Snippet 1: Course Recommendation System using GraphSAGE

In this section, we present the first code snippet, which demonstrates the implementation of a Course Recommendation System utilizing GraphSAGE. This system leverages graph-based learning to recommend courses based on shared skills and user preferences.

The following code snippet outlines the key steps involved in building and visualizing the recommendation graph:

```
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

import torch
from torch_geometric.data import Data
from torch_geometric.utils import from_networkx
import torch.nn.functional as F
from torch_geometric.nn import SAGEConv
from torch.nn.functional import cosine_similarity
```

```python

# Load the dataset
course_dataset = pd.read_csv("/content/try_file.csv")

# Initialize the graph
G = nx.Graph()

# Add nodes (courses) to the graph with attributes
for i in range(len(course_dataset['CourseID'])):
    G.add_node(course_dataset['CourseID'][i],
               name=course_dataset['course'][i],
               skills=course_dataset['skills'][i],
               rating=course_dataset['rating'][i],
               level=course_dataset['level'][i],
               certificate_type=course_dataset['certificatetype'][i],
               duration=course_dataset['duration(Months)'][i])

# Add edges based on shared skills, difficulty level, and certificate type
for i in range(len(course_dataset['CourseID'])):
    for j in range(i+1, len(course_dataset['CourseID'])):
        skills_i = set(skill.strip() for skill in course_dataset['skills'][i].split(','))
        skills_j = set(skill.strip() for skill in course_dataset['skills'][j].split(','))
        common_skills = skills_i.intersection(skills_j)

        weight = len(common_skills)
        if weight > 2:
            G.add_edge(course_dataset['CourseID'][i],
                       course_dataset['CourseID'][j],
                       weight=weight)

# Visualize the graph
plt.figure(figsize=(30, 20))
pos = nx.spring_layout(G, seed=42)
nx.draw(G, pos, with_labels=True,
        labels=nx.get_node_attributes(G, 'name'),
        node_color='skyblue', node_size=2000,
        font_size=10, edge_color='gray', linewidths=0.5)
edge_labels = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)
plt.title("Course␣Recommendation␣Graph")
plt.show()

# Create edge index and node features for PyTorch Geometric
edge_index = torch.tensor(list(G.edges), dtype=torch.long).t
```

```
    ().contiguous()
node_features = torch.tensor(np.array([
    course_dataset['rating'].values,
    course_dataset['level'].values,
    course_dataset['certificatetype'].values,
    course_dataset['duration(Months)'].values
]).T, dtype=torch.float)

# Convert NetworkX graph to PyTorch Geometric graph
data = from_networkx(G)
data.x = node_features

# Add edge weights
edge_weight = [G[u][v]['weight'] for u, v in G.edges]
data.edge_attr = torch.tensor(edge_weight, dtype=torch.float)

print(data)
print(f"Node␣features␣shape:␣{node_features.shape}")
print(f"Node␣features␣sample:␣{node_features[:5]}")

# GraphSAGE Model
class GraphSAGE(torch.nn.Module):
    def __init__(self):
        super(GraphSAGE, self).__init__()
        self.conv1 = SAGEConv(data.num_node_features, 16)
        self.conv2 = SAGEConv(16, 4)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.conv2(x, edge_index)
        return x

model = GraphSAGE()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

# Generate course embeddings
course_embeddings = model(data)
print(f"course_embeddings␣shape:␣{course_embeddings.shape}")

# Example: User preferences
user_rating = 4.8
user_level = 2
user_certificate_type = 3
user_duration = 1
user_features = torch.tensor([user_rating, user_level,
    user_certificate_type, user_duration], dtype=torch.float)

```

```python
# Compute cosine similarity between user preferences and
    course embeddings
similarities = cosine_similarity(user_features,
    course_embeddings)
print(similarities)

# Recommend top courses
top_k = 4
recommended_courses_idx = torch.topk(similarities, top_k).
    indices
recommended_courses = [G.nodes[n]['name'] for n in
    recommended_courses_idx.tolist() if n in G.nodes]
print(f"Recommended␣courses:␣{recommended_courses}")
```

Listing 8.1: Course Recommendation System using GraphSAGE

Chapter 9

TESTING

Testing is the process of executing a program with the intent of finding errors. It involves identifying defects, ensuring the software functions as intended, and validating that the system meets the specified requirements. Testing can be done at various stages of software development to ensure the quality and reliability of the product. [2]

9.1 Test Case Design

Test case design is the process of creating a set of inputs, execution conditions, and expected results that will allow thorough testing of a program. A well-designed test case ensures that all critical functionalities are tested, including edge cases and scenarios that may lead to unexpected behavior. The key objectives of test case design include:

(i) **Ensuring coverage of the entire system**: Every part of the code should be tested at least once to uncover defects.

(ii) **Finding defects early**: Test cases should be designed to identify potential errors as early in the development process as possible.

(iii) **Minimizing redundancy**: Tests should be efficient, avoiding duplication, and covering as many scenarios as possible without overlap.

In this chapter, we designed test cases based on control flow analysis, functional requirements, and edge cases to ensure complete testing of the Rule-Based recommendation module.

Test Case ID	Test Case Description	Input	Expected Output
TC-EC-01	Valid minimum support and confidence	min_support = 0.5, min_confidence = 0.7, user_preferences = {Python}	Recommendations based on user preferences (e.g., {Machine Learning})
TC-EC-02	Valid user preferences with multiple courses	min_support = 0.5, min_confidence = 0.7, user_preferences = {Python, Data Science}	Recommendations based on the intersection of user preferences and frequent itemsets
TC-EC-03	Valid minimum support, no user preferences	min_support = 0.5, min_confidence = 0.7, user_preferences = {}	Recommendations from popular courses based on the dataset
TC-EC-04	Valid minimum support and confidence, one item	min_support = 0.5, min_confidence = 0.7, user_preferences = {ML}	Recommendations based on user preferences (e.g., {Deep Learning})
TC-EC-05	Invalid minimum support (below range)	min_support = -0.1, min_confidence = 0.7, user_preferences = {Python}	Error message indicating invalid minimum support value
TC-EC-06	Invalid minimum confidence (below range)	min_support = 0.5, min_confidence = -0.5, user_preferences = {Python}	Error message indicating invalid minimum confidence value
TC-EC-07	Invalid user preferences (not a set)	min_support = 0.5, min_confidence = 0.7, user_preferences = "Python"	Error message indicating invalid input type for user preferences
TC-EC-08	Invalid user preferences (exceeds limit)	min_support = 0.5, min_confidence = 0.7, user_preferences = {Python, ML, Data Science, AI, Web Development}	Recommendations based on a valid set, though user preferences exceed a predefined size limit (if applicable)
TC-EC-09	Valid minimum support but no frequent itemsets found	min_support = 0.9, min_confidence = 0.7, user_preferences = {Python}	No recommendations available due to lack of frequent itemsets
TC-EC-10	Invalid empty transactions dataset	transactions = {}, min_support = 0.5, min_confidence = 0.7, user_preferences = {Python}	Error message indicating no data available for processing

Table 9.1: Equivalence Class Testing for Course Recommendation Module

9.2 Flow Graph

The control flow of a program can be analyzed using a graphical representation known as a flow graph. A flow graph is a directed graph where the nodes represent either entire statements or fragments of a statement, and the edges represent the flow of control between these statements. Each node can represent a block of code or an individual statement, while each edge indicates the possible transfer of control between two nodes.

For the given program, the flow graph was derived from the control structure of the code. Each decision point (conditional statement) creates branches in the flow graph, leading to multiple paths that the program can take during execution. The flow graph is essential in determining the program's complexity and identifying independent paths for testing. [3]

9.3 Basis Path Set

The basis path set is a set of independent paths through the flow graph that provides a basis for defining the program's structure. Each path in the basis set is a unique combination of decisions and loops, covering all possible execution paths. By selecting a minimal set of linearly independent paths, we can ensure that every statement and condition in the program is tested at least once.

For the given flow graph, the following basis path set was identified:

1. Path 1: $(1, 2, 3, 7, 8, 14, 15, 16, 21)$
2. Path 2: $(1, 2, 3, 4, 5, 6, 3, 7, 8, 14, 15, 16, 21)$
3. Path 3: $(1, 2, 3, 4, 6, 7, 8, 14, 15, 16, 21)$
4. Path 4: $(1, 2, 3, 4, 5, 6, 7, 8, 9, 13, 8, 14, 15, 16, 21)$
5. Path 5: $(1, 2, 3, 7, 8, 9, 10, 11, 12, 9, 13, 8, 14, 15, 16, 21)$
6. Path 6: $(1, 2, 3, 7, 8, 9, 10, 12, 9, 13, 8, 14, 15, 16, 21)$
7. Path 7: $(1, 2, 3, 7, 8, 9, 10, 11, 12, 9, 13, 8, 14, 15, 16, 17, 18, 19, 16, 21)$
8. Path 8: $(1, 2, 3, 7, 8, 9, 10, 11, 12, 9, 13, 8, 14, 15, 16, 17, 19, 16, 21)$

Each of these paths is independent, meaning they include unique combinations of decisions and loops, ensuring thorough testing of the control flow.

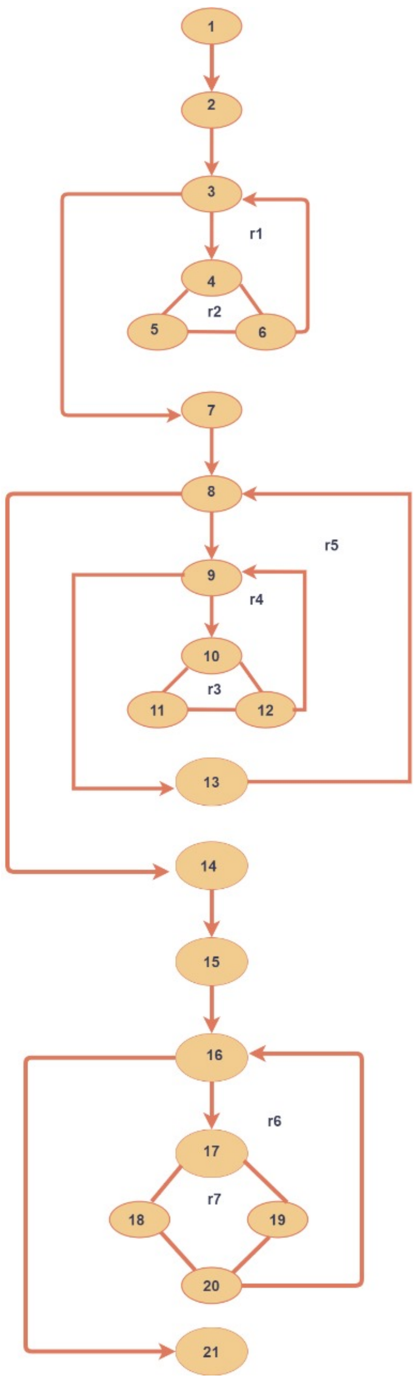

Figure 9.1: Flow Chart of the Rule-Based Recommendation Module

9.4 Cyclomatic Complexity

Cyclomatic complexity is a software metric used to measure the complexity of a program. It is defined as the number of linearly independent paths in the program. This metric helps to determine the number of test cases needed to cover all the paths in the program. Cyclomatic complexity is calculated using the formula:

$$CC = E - N + 2P$$

Where:

- E is the number of edges (transitions between nodes),
- N is the number of nodes (code blocks or statements),
- P is the number of connected components (for most programs, $P = 1$).

For the given flow graph:

- Edges (E) = 27
- Nodes (N) = 21
- Connected Components (P) = 1

Substitute these values into the formula:

$$CC = 27 - 21 + 2(1) = 8$$

The cyclomatic complexity of the given program is 8, meaning there are 8 independent paths. These paths represent the minimum number of test cases required to ensure complete path coverage during testing.

Bibliography

[1] K.K. Aggarwal and Y. Singh. *Software Engineering*. New Age International (P) Limited, 2005.

[2] Canva Pty Ltd. Canva - online design and visual communication tool. https://www.canva.com, 2024.

[3] Diagrams.net. Diagrams.net - online diagramming tool (formerly draw.io). https://www.diagrams.net, 2024.

[4] William Grosky and Terry Ruas. Data science for software engineers. In Roger Pressman and Bruce Maxim, editors, *Software Engineering: A Practitioner's Approach 9th Edition*, pages 629–638. McGraw Hill, New York, NY, 2019. Appendix 2.

[5] P. Jalote. *An Integrated Approach to Software Engineering*. Texts in Computer Science. Springer, 2005.

[6] OpenAI. Chatgpt: Language model by openai. https://chat.openai.com, 2024.

[7] R.S. Pressman and B.R. Maxim. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education, 2019.

[8] Stack Exchange Inc. Stack overflow - where developers learn, share, & build careers. https://stackoverflow.com, 2024.

[9] Wikipedia contributors. Software engineering - wikipedia. https://en.wikipedia.org/wiki/Software_engineering, 2024.