Project Proposal

AI-Driven Automated Requirements Engineering System

Members- Devanshi Jain,Sreyas Sharma,Shivang Mangal,Shreyash Shivhare

Mentor- Mitali Chugh Ma'am

1. Introduction

Requirements Engineering (RE) is a critical phase in the Software Development Life Cycle (SDLC), involving the identification, analysis, documentation, and prioritization of stakeholder requirements. In large-scale and agile-driven projects, stakeholders communicate requirements through emails, chat platforms, documents, and meetings, resulting in unstructured, scattered, and continuously evolving data.

The significance of Requirements Engineering lies in its direct impact on project success. Studies and industry experience consistently show that unclear, incomplete, or misunderstood requirements are among the primary causes of project delays, cost overruns, and system failures. A well-executed RE process ensures that stakeholder expectations are accurately captured, validated, and translated into implementable system specifications, thereby reducing rework and improving overall software quality.

However, modern software development environments pose several challenges to traditional RE practices. Requirements are no longer documented solely in structured specification documents; instead, they are dispersed across informal communication channels such as emails, instant messaging applications, collaborative tools, and meeting transcripts. This leads to multiple issues, including information loss, redundancy, ambiguity, and conflicting requirements. Additionally, in agile and iterative development settings, requirements frequently evolve, making manual tracking and prioritization increasingly complex and inefficient.

Current RE practices rely heavily on manual effort, requiring analysts to read, interpret, and consolidate large volumes of unstructured text. This process is time-consuming, error-prone, and does not scale effectively as the number of stakeholders and communication sources increases. Moreover, manual approaches struggle to provide timely insights such as requirement similarity, priority assessment, and high-level summaries needed for effective stakeholder decision-making.

To address these challenges, this project proposes an AI-driven, desktop-based automated requirements engineering system that leverages Natural Language Processing (NLP) and Machine Learning (ML) techniques. The proposed system automatically extracts requirement statements from diverse unstructured sources, classifies them into functional and non-functional categories, clusters semantically similar requirements to reduce redundancy, prioritizes them based on importance indicators, and generates concise summaries for improved stakeholder understanding. By automating key RE activities, the system aims to significantly reduce manual effort, improve accuracy and consistency, and provide a scalable and industry-ready solution for managing requirements in communication-heavy software projects.

2. Literature Review

Research on automating requirements engineering using NLP and AI has gained momentum due to the increasing volume of unstructured stakeholder data.

Early approaches primarily relied on rule-based systems, using linguistic patterns such as modal verbs (“shall”, “must”, “should”) to identify requirement statements. While effective for formal documents, these methods struggled with informal or conversational text.

Subsequent studies introduced machine learning-based classification techniques to categorize requirements into functional and non-functional types. These approaches improved scalability but required labeled datasets and often lacked domain adaptability.

Recent research explored clustering and topic modeling techniques (such as TF-IDF with K-Means and LDA) to group semantically similar requirements, enabling redundancy detection and thematic analysis. Additionally, Named Entity Recognition (NER) has been used to extract actors, system components, and actions, improving structural clarity.

However, most existing solutions:

- Focus primarily on static documents
- Do not support live communication channels
- Lack integrated, deployable architectures
- Do not provide end-to-end automation

The proposed system builds upon these foundations by offering a unified, deployable, and industry-oriented AI-driven requirements engineering pipeline.

3. Objectives of the Proposed System

The primary objectives of the system are:

- Automatically extract software requirements from unstructured text sources
- Classify requirements into functional and non-functional categories
- Cluster similar or related requirements
- Prioritize requirements based on importance indicators
- Generate concise summaries for stakeholder understanding
- Reduce manual effort and improve accuracy in the RE phase

4. Development Methodology

4.1 Agile Scrum Approach

- Development is divided into multiple time-boxed sprints
- Each sprint delivers a working increment (e.g., extraction, clustering, prioritization)
- Continuous stakeholder feedback is incorporated
- Agile is chosen due to:
    - Evolving AI models
    - Experiment-driven improvements
    - Changing stakeholder requirements

4.2 Kanban for Task Management

Kanban principles are used alongside Scrum to:

- Track tasks visually
- Manage AI experimentation workflows
- Improve development transparency

5. System Architecture Overview

Architectural Layers:

1. Desktop Application Layer (Frontend)
2. Backend Orchestration Layer
3. AI / NLP Processing Layer
4. Data Handling and Integration Layer

5. Frontend Design and Methodology

(Electron + React + Vite)

5.1 Electron-Based Desktop Application

- The system is implemented as a cross-platform desktop application using Electron.
- Enables deployment on Windows, Linux, and macOS
- Provides:

    - Native-like UI
    - Local processing capabilities
    - Secure communication with backend services

5.2 User Interface with React and Vite

- React.js is used for its:
    - Component-based architecture
    - Efficient state management
    - Reusability
- Vite is used as the build tool to ensure:
    - Faster development builds
    - Optimized production bundling

5.3 Frontend Capabilities

The UI allows users to:

- Upload documents (PDFs, text files)
- Connect communication platforms (emails, chats)
- Trigger AI analysis
- View:
    - Extracted requirements
    - Classified categories
    - Clustered groups
    - Priority levels
    - Summarized insights

6. Backend Methodology

(Spring Boot – Orchestration Layer)

Responsibilities of the Backend:

- Handle API requests from the Electron frontend
- Manage authentication and authorization
- Fetch data from external communication platforms
- Validate and preprocess textual data

- Communicate with the AI/NLP microservice
- Send structured responses to the frontend

7. Data Collection and Integration Layer

7.1 Email Integration

- Emails are fetched using:
  - IMAP protocol
  - Gmail API

- Features:
  - Secure authentication
  - Extraction of email body text
  - Processing of document attachments (PDF, DOCX)

7.2 Text Messages Integration

- Messages are accessed via:
  - Supported messaging APIs
  - Exported logs (where APIs are unavailable)
- Metadata such as timestamps and sender information are preserved
- Privacy-sensitive fields are handled securely

7.3 Telegram Integration

- Implemented using:
  - Telegram Bot API
  - Telegram client libraries
- Authorized access enables:
  - Message retrieval from chats or groups
  - Text normalization and aggregation

7.4 Unified Text Format

All collected data is:

- Cleaned
- Normalized

- Converted into a common text representation
  before being sent to the AI layer.

8. AI and NLP Methodology

(Flask-Based Python Microservice)

8.1 Flask NLP Microservice

- All AI and NLP logic is implemented as a Python microservice
- Flask is used due to:
  - Lightweight design
  - Easy REST API exposure
  - Seamless ML integration

8.2 NLP Techniques Employed

a) Text Preprocessing

- Noise removal
- Sentence segmentation
- Tokenization
- Text normalization

b) Requirement Extraction

- Rule-based linguistic patterns (e.g., *shall, must, should*)
- Named Entity Recognition (NER) to identify:
  - Actors
  - Actions
  - System components

c) Requirement Classification

- Functional vs Non-Functional classification
- Machine learning classifiers using NLP features

d) Requirement Clustering

- TF-IDF vectorization
- K-Means clustering
- Topic modeling as future enhancement

e) Requirement Prioritization

- Keyword-based importance
- Frequency-based scoring
- Requirement type weighting

f) Text Summarization

- Extractive summarization
- Generation of concise stakeholder-friendly summaries

9. Evaluation Strategy

To validate system effectiveness, multiple evaluation strategies are employed.

9.1 Requirement Extraction Evaluation

- Extraction accuracy
- Precision, recall, and F1-score

9.2 Classification Evaluation

- Precision, recall, and F1-score per class
- Confusion matrix analysis
- Macro and micro averaging

9.3 Clustering Evaluation

- Silhouette score
- Intra-cluster and inter-cluster similarity
- Expert-based qualitative validation

9.4 Prioritization Evaluation

- Rank correlation with expert priorities
- Stakeholder feedback surveys

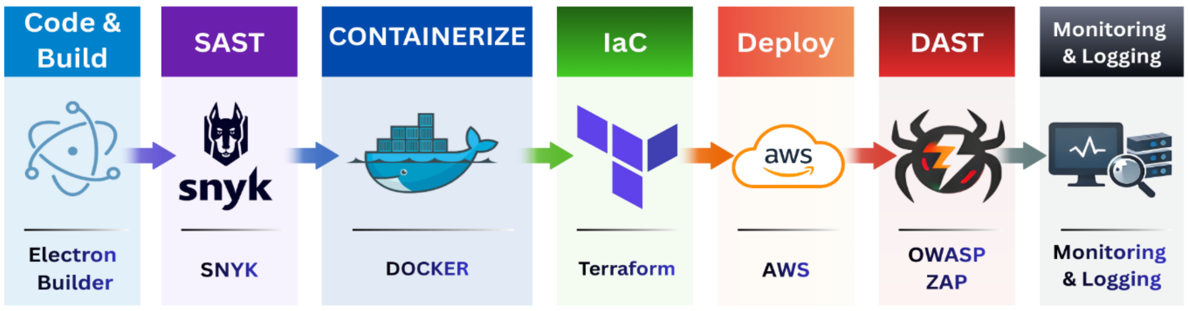
9.5 System-Level Evaluation

- End-to-end latency
- Scalability testing
- Usability assessment

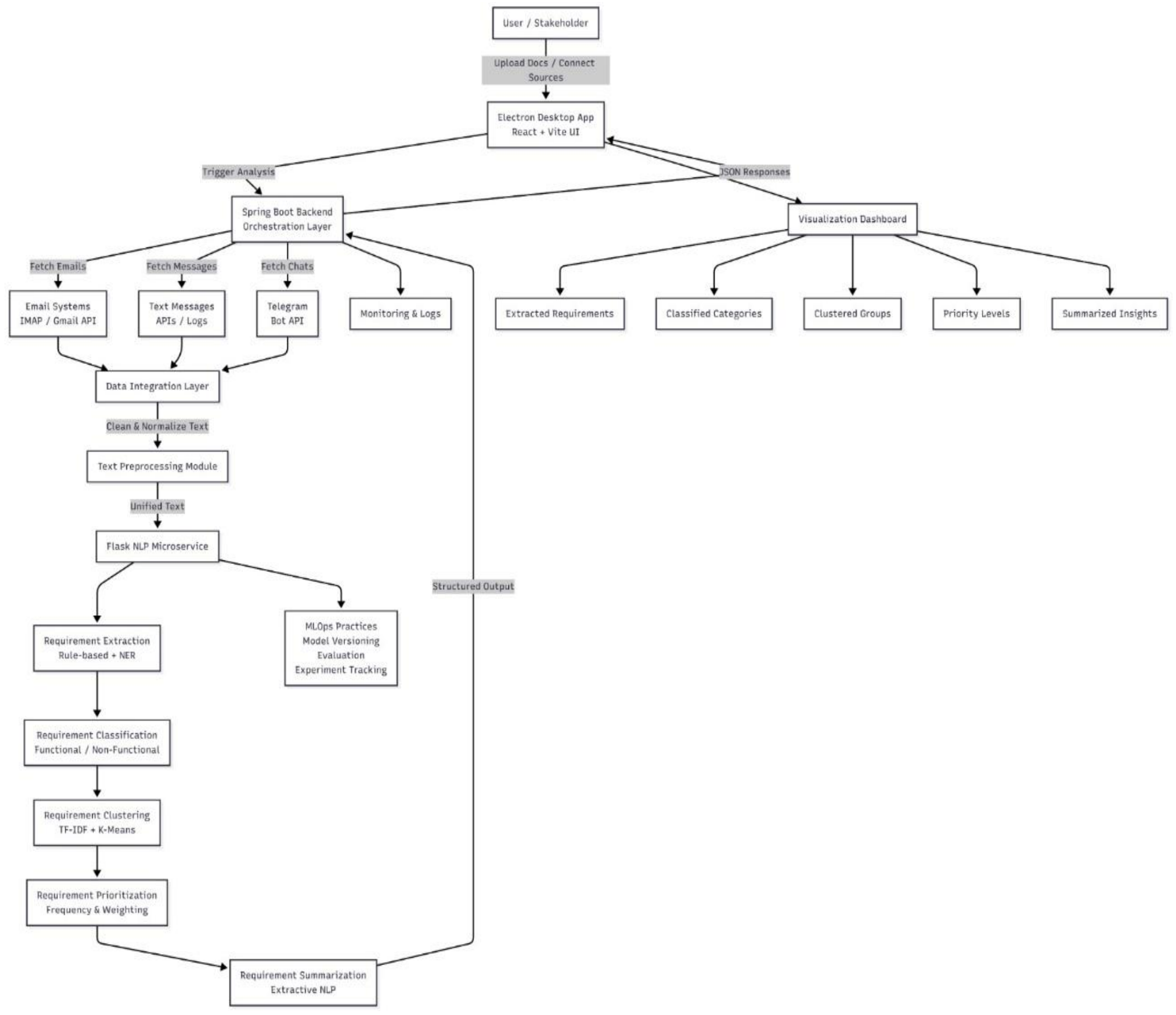
10.System Workflow

1. User interacts with the Electron–React UI
2. Backend fetches and preprocesses data
3. Cleaned text is sent to Flask AI service
4. AI service performs:
    - Extraction
    - Classification
    - Clustering
    - Prioritization
    - Summarization
5. Results are returned to frontend for visualization

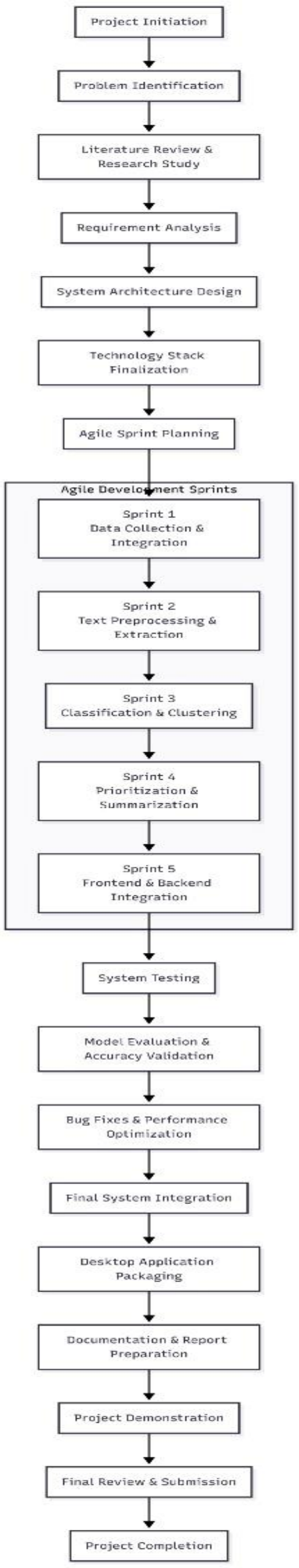
11.CI/CD Flow



12.Flow Diagram


User / Stakeholder
Upload Docs / Connect Sources
Electron Desktop App React + Vite UI
Trigger Analysis
JSON Responses
Spring Boot Backend Orchestration Layer
Visualization Dashboard
Fetch Emails
Fetch Messages
Fetch Chats
Email Systems IMAP / Gmail API
Text Messages APIs / Logs
Telegram Bot API
Monitoring & Logs
Extracted Requirements
Classified Categories
Clustered Groups
Priority Levels
Summarized Insights
Data Integration Layer
Clean & Normalize Text
Text Preprocessing Module
Unified Text
Flask NLP Microservice
Structured Output
Requirement Extraction Rule-based + NER
MLOps Practices Model Versioning Evaluation Experiment Tracking
Requirement Classification Functional / Non-Functional
Requirement Clustering TF-IDF + K-Means
Requirement Prioritization Frequency & Weighting
Requirement Summarization Extractive NLP

13.Development Cycle


Project Initiation
Problem Identification
Literature Review & Research Study
Requirement Analysis
System Architecture Design
Technology Stack Finalization
Agile Sprint Planning
Agile Development Sprints
Sprint 1 Data Collection & Integration
Sprint 2 Text Preprocessing & Extraction
Sprint 3 Classification & Clustering
Sprint 4 Prioritization & Summarization
Sprint 5 Frontend & Backend Integration
System Testing
Model Evaluation & Accuracy Validation
Bug Fixes & Performance Optimization
Final System Integration
Desktop Application Packaging
Documentation & Report Preparation
Project Demonstration
Final Review & Submission
Project Completion

14.Future Scope

The system can be extended to include:

- Real-time message ingestion
- Ambiguity detection in requirements
- Conflict and dependency analysis
- Stakeholder-aware prioritization
- Requirement Traceability Matrix (RTM)
- Transformer-based NLP models (BERT, etc.)

15.Conclusion

The proposed system delivers a practical, scalable, and industry-aligned solution to automate the Requirements Engineering phase using AI. By combining:

- Electron-based desktop UI
- Spring Boot orchestration backend
- Flask-based NLP microservices
- Agile Scrum development methodology

the solution effectively addresses the challenges of modern, communication-heavy software projects.