Requirement Based Diagram Generator

Final Year Project

Session 2021-2025

A project submitted in partial fulfilment of the
COMSATS University Degree
of
BS in Computer Science / Software Engineering (CUI)

Department of Computer Science

COMSATS University Islamabad, Lahore Campus

30 Dec 2024

Project Detail

Type (Nature of project)	[ ] **D**evelopment [ ] **R**esearch [✓] **R&D**
Area of specialization	

Project Group Members

Sr.#	Reg. #	Student Name	Email ID	*Signature
(i)	Sp21-Bse-035	Muhammad Nouman	Sp21-bse-035@cuilahore.edu.pk	Nouman
(ii)	Sp21-Bse-003	Abdullah Hashmi	Sp21-bse-003@cuilahore.edu.pk	Abdullah
(iii)	Sp21-Bse-048	Javeria Shakoor	Sp21-bse-048@cuilahore.edu.pk	Javeria

Plagiarism Free Certificate

This is to certify that, I am **Muhammad Nouman** S/D/o **Muhammad Ashiq**, group leader of FYP under registration no **CIIT/Sp21-Bse-035/LHR** at Computer Science Department, COMSATS Institute of Information Technology, Lahore. I declare that my FYP proposal is checked by my supervisor and the similarity index is_________% that is less than 20%, an acceptable limit by HEC. Report is attached herewith as Appendix A.

Date: **15th May 2024** Name of Group Leader: **Muhammad Nouman** Signature: Nouman

Name of Supervisor: Ma'am Mamoona Tassaduq		Co-Supervisor : Ma'am Humera Afzal	
Designation:	___________	Designation:	________________________
Signature:	___________	Signature:	________________________

Abstract

Our project's main aim is to develop a system that allows the user to generate the system architecture diagram along with other UML diagrams. The diagrams will be based on the system requirements provided by the user. The technology of the modern day is very fast-paced and constantly developing. Due to this, the development process is forced to speed up and suffers in quality. This is why system architecture and UML diagrams are of the utmost importance because due to them, quality products are made. Hence, we aim to bridge the gap between the user and system architects, and in doing so, we streamline the whole process.

The core functionality of the system revolves fully around taking the user requirements from the user through a user-friendly interface. Our project aims to provide flexibility and ease of communication hence we will be taking requirements from the user in natural language form. The system will prase these requirements and analyze the key elements in the requirements such as functionalities, context, constraints, and dependencies within them.

The next involves the usage of machine learning algorithms that will match the requirements and their finding to suitable components. These components will represent the modules of the system thus generating an optimized system architecture. We will also consider factors like scalability, performance, and cost-effectiveness to ensure the system meets the user's needs in an effective manner.

Once the diagrams have been generated, they will be represented in an editable environment where the user is able to change the diagram, add new components etc., to their liking. Feedback from the user will also be noted and thus, this will enable iterative improvement and refinements.

From this project, we want the user to be able to clearly articulate their needs in the form of requirements and turn them into visual representations tailored to their needs. Our work and findings hope to contribute in the development process of systems related to various domains.

Table of Contents

List of Tables

List of Figures

1 Introduction

It has been seen that when engineers are dealt with translating user needs into a well-defined system that reflects those requirements, the process seems to be quite complex and has a lot of chance for error. Moreover, developing these diagrams is crucial in order to make a successful and quality product. Since this process has the potential for a lot of errors and is complex in nature, it requires a lot of time to get it right. This is where our project comes in, we are proposing the development of a Requirement-Based System Architecture Generator hence automating and streamlining the tedious process of requirement analysis and making system diagrams out of them.

1.1 Objectives:

The project aims to develop a system that is capable of processing/understanding/contextualizing a given set of requirements and generating multiple visual models such as System architecture along with other UML diagrammatic representations such as use cases and activity diagram.

The main objectives of the proposed software include:

i. **Development of user-friendly interface:** To make System accessible and easy to use, the project aims to develop an appealing and user-friendly interface that is highly easy to use.

ii. **Incorporate Natural language processing techniques(NLP):** Utilization of NLP algorithms is a primary focus of the project. The process will be done to analyze user requirements, extract key information, and understand the context.

iii. **Generating Diagram through Machine Learning:** Adopting Machine Learning algorithms to build suitable components that are well optimized in generating system architecture and other diagrams.

iv. **Representation of Diagram:** Provide visual representations and documentation of the generated architectures for better understanding and communication.

1.2 Problem Statement

Automate the process of requirement analysis and system architecture generation in software engineering. Assist the development team by increasing efficiency, and accuracy and mitigating costs in the design phase.

1.3 Assumptions & Constraints

1.3.1 Assumptions

The project aims to tackle the task of generating system architecture from given system requirements. Although this is a hefty task we expect to fulfil this by assuming the following.

- Assuming that User understands the need and importance of software architecture andits role in the software development life cycle(SDLC).
- Assuming that users are familiar with the terminology and notations imposed by our software regarding system architectures along with their elements and components.
- Assuming that users are comfortable working in a desktop application.
- Assuming that the provided dataset of requirements does not contain any contradictions within therequirements
- Assuming that the user will follow the guidelines during the use of software.

1.3.2 Constraints

The following constraints must be considered to ensure the success of the project.

- The projects that we are using to train the model are different in nature in terms of their domain and their system architecture.
- Some projects have an insufficient number of requirements reflecting the modules.
- The inability to conduct extensive testing poses a notable constraint.
- Providing an intuitive user interface to users is very important as that directly relates to one of the core functionalities.

1.4 Project Scope

1.4.1 Included:

- ✓ Development of an automated system for generating diagrams from user-provided requirements.
- ✓ Creation of a user-friendly web-based interface for inputting requirements and interacting with System.
- ✓ Integration of NLP libraries for tokenizing and parsing user requirements.
- ✓ Development of machine learning models to optimize system architectures.
- ✓ Implementation of algorithms for automatically generating activity and use case diagrams.
- ✓ Integration of visualization tools for visualizing generated architectures and diagrams.
- ✓ Automated generation of comprehensive documentation for architectures.
- ✓ Adoption of an agile development methodology for iterative development.

1.4.2 Not Included:

- Extensive testing beyond basic validation.
- Deployment in a production environment.
- Ongoing maintenance and support beyond the project timeframe.
- Advanced NLP or ML techniques.
- Extensive user testing with diverse user groups.

2 Requirements Analysis

2.1 Literature review / Existing system study

2.1.1 Literature review

In the area of research, many researches have made significant advancements in systems that generate automated UML diagrams from user requirements. For instance,

1. **Zhong et al**. (2023) [1] proposed an approach utilizing NLP techniques to generate Systems Modelling Language (**SysML**) diagrams from unstructured text, streamlining the requirements analysis process.
2. Similarly, **Deeptimahanti and Babar** (2009) [2] developed UMGAR which stands for UML Model generator from analysis of requirements i.e a handsome tool leveraging NLP and domain ontology techniques for the development of various UML models from user requirements.
3. **Priyanka More and Rashmi Phalnikar** (2012) [3] introduced "**RAPID**," which is tool for automatically generating UML diagrams from user requirements and specifications and whose main aim is to focus on enhancing the efficiency and accuracy of requirements analysis.
4. **Richa** et al. (2014) [4] proposed a structured approach for converting user requirements into some intermediate structural representation i.e **frames** and then, using those intermediate structures to transform into behavioural UML models..
5. **MYL Gamage** (2023) [5] proposed an automated application which employs a novel pipeline for information extraction and diagram generation, incorporating a custom Named Entity Recognition Model and a **Masked BERT Relation Classifier** along with shallow heuristics and a **knowledge graph** for diagram relation mapping.

2.1.2 Existing Studies

Recent advancements have led to innovative applications facilitating automated diagram generation from textual descriptions. For instance,

1) **Eraser Diagram GPT (diagramgpt):**[1] A tool powered by Eraser, allowing users to generate diagrams from natural language descriptions effortlessly. It's ideal for quickly visualizing concepts without manual drawing.

[1] DiagramGPT – Powered by Eraser

2) **Mermaid.js[2]:** A JavaScript library for creating diagrams and flowcharts from textual descriptions using a simple syntax. It's versatile, supporting various diagram types like flowcharts, sequence diagrams, and Gantt charts, and can be integrated into web applications or documentation systems.
3) **PlantUML[3]:** A text-based tool for generating UML diagrams with a human-readable syntax. Users write UML diagrams in plain text, and PlantUML automatically renders them into visual diagrams. It supports a wide range of UML diagram types and can be used standalone or integrated into other platforms.

2.2 Stakeholder List

2.2.1 User:

User-targeted are software engineers who have knowledge of software system diagrams, their notations, and their meanings.

2.2.2 Developing Team:

The developing team consists of three software engineering students under the guidance of a Professor.

2.3 Requirement Elicitationa

2.3.1 Functional Requirements

Export Sub-Module

User Registration (FR01):

Table 1: Functional Requirements

Export to Multiple Formats (FR1):

FR1-01	System will enable users to export generated diagrams in a variety of file formats, including but not limited to XML, PNG, Json, Mermaid.js.
FR1-02	Users will have the flexibility to choose the most appropriate format for their specific requirements, as outlined above.

Feedback on Export Completion (FR2):

FR2-01	System will deliver appropriate feedback to User upon successful completionof the export process.
FR2-02	System shall display confirmation messages indicating that Diagrams havebeen exported successfully and are ready for opening or sharing.

[2] https://mermaid.js.org/intro/

[3] https://www.plantuml.com/plantuml/uml/SyfFKj2rKt3CoKnELR1Io4ZDoSa70000

Download Files (FR03):

FR03-01	System will allow the user to download the specific file he choose he want to export.

XML Architecture Generator:

Input Requirements (FR04):

FR04-01	System shall extract the elements from the parsed requirements.
FR04-02	System shall identify the dependencies, interactions and constraints from the requirements to establish the relationships between different components and modules.

XML Representation (FR05):

FR05-01	System shall define constraints, data types and structure schema for representing architecture elements, relationships and attributes using XML.
FR05-02	System shall map extracted requirements and relationships onto the defined schema.
FR05-03	System shall establish correspondences between system requirements and XML elements, relationships and attributes.
FR05-04	System shall generate JSON documents based on the mapped architecture model.
FR05-05	System shall validate the generated JSON documents against the defined schema.
FR05-06	System shall support customization and extension of the schema to accommodate domain-specific and requirements and architectural patterns if any.

XML Architecture Representation/Diagram Generator:

Input Components (FR06):

FR06-01	System shall accept XLSX documents containing architecture Requirement as input.
FR06-02	System shall parse the document to extract components, modules, sub-modules, entities, relationships and attributesnetwork.

Diagrammatic representation (FR07):

FR07-01	System shall use libraries to map the extracted network of architecture modelinto a graphical representation.

Editor / Canvas

User Interface (FR08):

FR08-01	System shall provide users with a user-friendly and intuitive interface thatcan be used for editing the generated diagram.
FR08-02	System shall allow User to zoom in and zoom out diagrams dependingupon the view they want.
FR08-03	System shall allow Users to drag and drop components and connectors.

Diagram Creating (FR09):

FR09-01	System shall also allow User to create diagrams from scratch.
FR09-02	System shall provide Users with predefined templates and architectural patterns that they will be able to customize (e.g., client-server, pipes and filters,layered, etc).
FR09-03	System shall provide User with a sidebar that has multiple shape optionssuch as rectangles, ellipses, hexagons, etc, and connectors such as arrows, lines,etc.

Diagram Customization/Editing (FR10):

FR10-02	System shall allow Users to write and edit text in any shape or outside ofthe shapes.
FR10-03	System shall allow User to delete and duplicate the components in Diagram.
FR10-04	System shall facilitate User with different fonts and text styles such asbold, italic, and underlined.

Diagram Handling (FR11):

FR11-01	System shall allow User to save their diagram locally to System.
FR11-02	System shall allow Users to delete Diagrams created.

Import Sub-Module

Import Natural Language Requirements (FR12):

FR12-01	System will allow users to import natural language Functional requirements.
FR12-02	System shall support multiple file formats for input file namely excel spreadsheets (.xlsx), and CSV files (.csv).
FR12-03	If a user attempts to upload a file format not specified above, System must promptly notify User that the requirement extraction may not be feasible from thisfile format.

Parse Imported Requirements (FR13):

FR13-01	System shall parse the imported document to extract functional requirements for subsequent processing.
FR13-02	If the document does not contain any functional requirements, System must notifyUser to either include functional requirements in the document or upload another file.

Validate Imported Requirements (FR14):

FR14-01	System shall conduct validation checks on the imported requirements to ensuretheir completeness and correctness.
FR14-02	System will promptly alert users to any identified issues or inconsistencies in the imported requirements and offer guidance for resolution.

User Confirmation (FR15):

FR15-01	System will provide users with the option to review and confirm the imported requirements before proceeding with diagram generation
FR15-02	Additionally, it will allow users to make adjustments or corrections to the imported requirements as necessary.

Customization Options (FR16):

FR16-01	System shall provide customization options for users to specify additionalmetadata or annotations associated with imported requirements.
FR16-02	Furthermore, it shall allow users to categorize, prioritize, or add tags to imported requirements for organizational purposes.

2.3.2 Non-Functional Requirements

Table 2: Non-Functional Requirements

Requirement Number	**Description**
NFR01	System must consistently respond within 2 seconds for all user interactions.
NFR02	System needs to be available at least 90% of the time to ensure reliability.
NFR03	The should undergo testing to ensure robustness and reliability.
NFR05	Version No. control using Git will streamline collaboration among developers and track
	changes effectively.
NFR06	System must log all errors with time stamps and detailed error messages.
NFR08	Detailed error logging and reporting facilitate efficient issue resolution, enhancingoverall system reliability.

2.3.3 Traceability Matrix

Table 3: Requirements Traceability Matrix

Req No	Requirement Name	Sequence No	Activity No	Use Case No	Business Needs, Opportunities, Goals, Objectives	Project Objective No	Resource Allocation	Test Case ID
FR01	Export to Multiple Formats	SD02	AD02	USD05	Enhance interoperability and usability by supporting multiple export formats	iv	Nouman	TC_15
FR02	Feedback on Export Completion	N/A	AD02	USD05	Ensure user satisfaction and confidence through export feedback	iv	Abdullah	TC_16
FR03	Download Files	SD01	N/A	US07	Enhance user engagement and interaction through seamless download files functionalities.	i	Abdullah, Javeria	TC_02
FR04	Input Requirement	SD01	AD02	USD14	Ensure reliability and accuracy of imported requirements	i, ii, v	Nouman	TC_03
FR05	JSON Representation	SD03	AD02	USD06	Streamline the process of generating XML-based architecture	iii	Abdullah	TC_05
FR06	Input Components	SD01	AD02	USD14	Enhance low coupling and high cohesion	iii	Javeria	TC_06
FR07	Diagrammatic representation	SD03	AD04	USD19	Facilitate collaboration and communication through visual representations	iv	Javeria, Abdullah	TC_07

FR08	User Interface	SD04	AD05	USD19	Enhance user engagement by providing intuitive editing capabilities	iv, v	Abdullah	N/A
FR09	Diagram Creating	N/A	AD04	USD03	Enhance usability and customization options for users	iii, iv	Javeria	TC_08
FR10	Diagram Customization/Editing	SD04	AD03	USD02	Enhance user satisfaction with interactions	iv	Abdullah	TC_09
FR11	Diagram Handling	SD04	N/A	USD02	Enhance usability by auutomatically saving Diagram	iv	Javeria	TC_10
FR12	Import Natural Language Requirements	SD01	AD02	USD01	Improve user experience by enabling import of natural language requirements	ii, iv	Nouman	TC_11
FR13	Parse Imported Requirements	SD01	AD02	USD01	Enhance efficiency by automating parsing of imported requirements	ii, iv	Nouman, Javeria	TC_12
FR14	Validate Imported Requirements	SD01	AD02	USD01	Ensure data quality and completeness by validating imported requirements	ii	Abdullah	TC_13
FR15	User Confirmation	N/A	AD02	N/A	Enhance user control and accuracy through user confirmation of imported requirements	v	Javeria	TC_14
FR16	CustomizationOptions	N/A	AD02	N/A	Improve user customization and control over exported diagrams	iv	Javeria	TC_15

2.4 Use case descriptions

Table 4: Use Case Descriptions

Use Case Name:	Import Natural Language Requirements
Use Case ID:	US01
Actors:	User, System
Summary:	Allowing User to import their desired system requirements in various formats.
Pre - Condition(s):	● User possesses a document that has System requirements. ● User has access to our system.
Post - Condition(s):	● The requirements have been successfully imported for System to generate the architecture.
Inputs:	A document containing System requirements in a supported format.
Outputs:	Successfully parsed requirements of System ready to be further processed.
Normal flow:	1. User chooses the "Import Requirements Document" option. 2. User then browses and uploads the desired document containingthe requirements. 3. System parses it, verifies it, and then extracts the functional requirements.
Exception:	If the document is not in the supported format or if the parsed document does not contain any requirement, System notifies User through a prompt indicating the exact error.

Use Case Name:	Diagram Customization and Editing
Use Case ID:	US02
Actors:	User
Summary:	System allows User to customize and edit the generated diagrams according to their need.
Pre - Condition(s):	● User has access to Diagram and the editing environment.
Post - Condition(s):	● Diagram will be customized according to preference of User.
Inputs:	User preference for customization (e.g., resizing, color changes, font changes).
Outputs:	Customized and edited diagram.
Normal flow:	1. User accesses the editing interface. 2. There User sees many options regarding color, size, text, connections, shapes, etc. 3. User selects the desired options and applies them to Diagram. 4. The changes are reflected in Diagram.
Exception:	None

Use Case Name:	Generate JSON diagrams
Use Case ID:	US04
Actors:	System
Summary:	JSON diagrams are generated based on the requirements mapped.
Pre - Condition(s):	● The document is uploaded. ● Requirements are properly mapped and diagrams are ready to be generated.
Post - Condition(s):	● JSON diagrams generated.
Inputs:	Mapped diagram model, Requirements
Outputs:	JSON diagram
Normal flow:	1. System receives the requirements and parses the information. 2. System then maps the requirements. 3. System receives the mapped diagram model ready to be used. 4. The JSON diagrams are generated. Diagrams generated are validated against the defined schema.
Exception:	If System fails to generate Diagram due to errors during mapping foundduring validation, System promptly notifies User.

Use Case Name:	Export to Multiple Formats
Use Case ID:	US05
Actors:	User, System
Summary:	Users export the generated diagram in various file formats.
Pre - Condition(s):	● Diagram is generated and is to the liking of User.
Post - Condition(s):	● Diagram is exported in the desired format.
Inputs:	User choice for export format.
Outputs:	Exported diagram.
Normal flow:	1. User clicks on the "Export Diagram" option. 2. User then selects the format desired for the export. 3. System then exports progress through a bar. 4. Once it's done, System notifies User that Diagram hasbeen exported according to the desired format.
Exception:	If the export procedure fails (due to any technicality), System notifies User and prompts them to try again.

Use Case Name:	Validate Generated JSON diagram
Use Case ID:	US06
Actors:	System
Summary:	System validates the generated JSON diagram defined against some schema.
Pre - Condition(s):	● JSON diagram is generated.
Post - Condition(s):	● JSON diagram is fully validated.
Inputs:	JSON diagram.
Outputs:	Validated JSON diagram.
Normal flow:	1. System receives the generated JSON diagram. 2. System then compares the structure and components against the defined schema. 3. Validation checks are performed and in case of valid architecture, System notifies User.
Exception:	If System fails to validate Diagram or finds some inconsistencies or discrepancies, System notifies User and highlights the fault.

Use Case Name:	Save Diagram
Use Case ID:	US07
Actors:	User
Summary:	Allowing User to save their diagram for access in the future
Pre - Condition(s):	● Diagram has been created/generated.
Post - Condition(s):	● Diagram will be saved and will be accessible in the future.
Inputs:	Diagram to be saved.
Outputs:	Saved diagram file location.
Normal flow:	1. User initiates the save operation by clicking on the “Save diagram” option. 2. System prompts User for a file location. 3. After selecting the file location, Diagram is saved with the appropriate file format.
Exception:	If the saving of Diagram fails due to some unexpected error, System notifies User.

Use Case Name:	Alert User
Use Case ID:	US11
Actors:	System
Summary:	An error that is generated is notified to User through an error prompt.
Pre - Condition(s):	● The error is generated due to any reason (technical error or errors caused by User).
Post - Condition(s):	● User is successfully notified of the error prompt.
Inputs:	Generated Error.
Outputs:	Error prompt
Normal flow:	1. An error is generated by System due to some technicality or user error. 2. System receives the error and generates a prompt that explainsthe error. 3. User is notified of the error prompt.
Exception:	N/A

Use Case Name:	Map Requirements
Use Case ID:	US14
Actors:	System
Summary:	The entities and their relationships extracted from requirements are mapped onto the XML components.
Pre - Condition(s):	● Entities and relationships are extracted from requirements. ● XML schema is defined.
Post - Condition(s):	● Mapped XML schema
Inputs:	Entities and their relationships, XML schema
Outputs:	Mapped XML model
Normal flow:	1. System receives the requirement 2. It parses them to extract entities and relationships. 3. System identifies XML schema components.
	4. It then maps those entities and relationships onto the components. 5. System saves the mapped model ready to be used. 4. The XML diagrams are generated based on that mapped model.
Exception:	If System fails to map any entity or relationship due to errors during entityor relationship extraction, System promptly notifies User.

Use Case Name:	Identify relationships and dependencies
Use Case ID:	US15
Actors:	System
Summary:	User inputs requirements and System processes them to identify relationships and dependencies.
Pre - Condition(s):	● Requirements are pre-processed.
Post - Condition(s):	● Identified relationships and dependencies in requirements.
Inputs:	Pre-processed requirements
Outputs:	Identified dependencies and relationships
Normal flow:	1. System receives the requirement. 2. It parses them and validates to see if requirements exist. 3. If validated requirements exists, it performs word embedding on tokenize requirements. 4. After that, it applies dependency parsing and masked entity recognition to extract relationships and dependencies. 5. It then evaluates the dependencies and relationships to extract the best one.
Exception:	If System fails to find validated requirements during parsing, it promptly
	notifies User.

Use Case Name:	Support Layout Algorithms
Use Case ID:	US16
Actors:	System
Summary:	System supports layout algorithms that analyses the spatial arrangementsof diagram components and re-arranges them at their best presentable positions.
Pre - Condition(s):	● A mapped diagram model is defined.
Post - Condition(s):	● Formatted diagrams are generated.
Inputs:	Mapped diagram model
Outputs:	Formatted diagrams
Normal flow:	1. System receives the mapped diagram model. 2. It analyses the spatial arrangement of diagram components. 3. It calculates the best positions for each components. 4. It then maps Diagram components onto the calculated positions. 5. System displays the formatted diagrams.
Exception:	If System fails to find any optimized layout, it then displays Diagramin its original layout and informs User of layout.

Use Case Name:	Extract elements/tokens
Use Case ID:	US17
Actors:	System
Summary:	System shall parse the input requirements and tokenize them.
Pre - Condition(s):	● Requirements are given as input.
Post - Condition(s):	● Requirements are tokenized and ready for word embedding.
Inputs:	Requirements
Outputs:	Requirements tokens
Normal flow:	1. System receives User requirements 2. With the help of natural language processing, it apply tokenization on requirements. 3. The tokens are then used for word embedding and dependency parsing to extract relationships and dependencies.
Exception:	If System fails to tokenize, it informs User of the layout.

Use Case Name:	Zoom In/Out
Use Case ID:	US19
Actors:	User
Summary:	User zooms in or out of Diagram according to the view requirement.
Pre - Condition(s):	● User is on the editing page.
Post - Condition(s):	● Diagram is zoomed in/out accordingly.
Inputs:	User initiation
Outputs:	Zoomed in/out diagram
Normal flow:	1. User selects the appropriate symbol to zoom in/out. **8** If User chooses to zoom in, the option selected is the plus sign else it is the minus sign.
Exception:	

Use Case Name:	Customize Elements
Use Case ID:	US20
Actors:	User
Summary:	The elements are customized and edited according to User's needs/liking.
Pre - Condition(s):	● Diagram is created.
Post - Condition(s):	● Diagram elements are edited accordingly.
Inputs:	Color, Text, Size information, diagram element
Outputs:	Edited Elements
Normal flow:	1. User selects the element they want to edit. 2. After selection, User selects the property they want to edit, this could be its size, font, or color information. 3. User inputs the updated values for the chosen properties. 4. The changes are reflected in Diagram elements.
Exception:	If User adds an invalid amount for size or font, System notifies Userwith the error prompt.

Use Case Name:	Delete Elements
Use Case ID:	US21
Actors:	User
Summary:	User deletes the element that they have selected.
Pre - Condition(s):	● Diagram is created.
Post - Condition(s):	● Diagram element is deleted.
Inputs:	User initiation, diagram element
Outputs:	Element Deletion
Normal flow:	1. User selects the element they want to delete. 2. After that they can either press the delete button or the “Del” key. 3. System deletes the respective element and the change is reflected in Diagram.
Exception:	N/A

Use Case Name:	Delete Diagram
Use Case ID:	US22
Actors:	User, System
Summary:	User deletes Diagram and the change is reflected in System.
Pre - Condition(s):	● Diagram is created.
Post - Condition(s):	● Diagram is deleted from System.
Inputs:	User initiation, Diagram
Outputs:	Deleted Diagram
Normal flow:	1. In the editor panel, User selects the option to delete Diagram. 2. After that System asks User to confirm the deletion. 3. On confirming Diagram is deleted and the change is reflected in System.
Exception:	Diagram is not deleted if User cancels the deletion process.

Use Case Name:	Give Feedback
Use Case ID:	US23
Actors:	System
Summary:	System provides User with necessary feedback in return of the completion of a process.
Pre - Condition(s):	● A process has been completed.
Post - Condition(s):	● User is notified of the completion with additional information.
Inputs:	N/A
Outputs:	N/A
Normal flow:	1. Whenever a process is completed, let that process be of successful import of requirements, export of diagram, deletion of an account, etc, System notifies User. 2. User is notified through a prompt giving information about the process that was completed.
Exception:	N/A

Use Case Name:	Review Requirements
Use Case ID:	US24
Actors:	User, System
Summary:	User reviews the requirements before letting System generatediagrams from it.
Pre - Condition(s):	● User has imported the requirements document.
Post - Condition(s):	● The reviewed requirements are fed to System.
Inputs:	Requirements document, User revisions
Outputs:	Revised requirements
Normal flow:	1. After the requirements are imported into System, System displays the requirements to User. 2. User reviews the requirements document. 3. If revisions are needed, User makes the necessary changes in the requirements. 4. User re-submitted the requirements which are now revised.
Exception:	N/A

Use Case Name:	Confirm Requirements
Use Case ID:	US25
Actors:	User
Summary:	After User has imported the requirements and revised them, User then confirms the requirements.
Pre - Condition(s):	● User has imported the requirements document. ● User has revised the requirements.
Post - Condition(s):	The revised requirements are fed into System.
Inputs:	Revised requirements.
Outputs:	System confirmation.
Normal flow:	1. User has imported the requirements document into System. 2. User has reviewed the requirements to their liking. 3. After that, User confirms that the requirements have been finalized. 4. The requirements are then fed into System for generation.
Exception:	In case of any issue in the requirement document or the revised document, System notifies User of the error, and the requirements are not fed into System.

Use Case Name:	Categorize Requirements
Use Case ID:	US26
Actors:	User
Summary:	User will categorize requirements to their liking in per se modules.
Pre - Condition(s):	● The requirement document is imported.
Post - Condition(s):	● The requirements are categorized.
Inputs:	Imported requirements
Outputs:	Categorized requirements
Normal flow:	1. Once User has imported the requirements, System allows User to categorize the requirements. 2. User can select specific requirements and assign them
	categories or modules. 3. The requirements are then categorized accordingly.
Exception:	N/A

Use Case Name:	Add Tags to requirements
Use Case ID:	US27
Actors:	User
Summary:	User adds tags to requirements to further classify them.
Pre - Condition(s):	● The requirements are imported.
Post - Condition(s):	● The requirements are tagged.
Inputs:	Imported requirements
Outputs:	Tagged requirements
Normal flow:	1. Once User has imported the requirements, System allows User to tag the requirements. 2. User can select specific requirements and tag them with valuesthat can further classify requirements. 3. The changes are reflected in System.
Exception:	N/A

Use Case Name:	Verify Format
Use Case ID:	US28
Actors:	System
Summary:	System shall verify the format of input file if it is a supported file formator not.
Pre - Condition(s):	● Requirements file is given as input.
Post - Condition(s):	● Confirmation of upload or Alert of file discard.
Inputs:	Requirements file
Outputs:	System notification
Normal flow:	1. System receives User requirements file. 2. It will check it the file format is supported or not. 3. If the file format is supported, it will show a notification of a successful upload.
Exception:	If System fails to verify the format, it informs User of the file discard.

Use Case Name:	Verify Requirements
Use Case ID:	US29
Actors:	System
Summary:	System shall parse the input file to verify if the validated functional requirements exist.
Pre - Condition(s):	● Requirements file is given as input.
Post - Condition(s):	● Confirmation of upload or Alert of file discard.
Inputs:	Requirements file
Outputs:	System notification
Normal flow:	1. System receives User requirements file. 2. It will parse that file to verify if valid functional requirements exist in the file or not. 3. If the file contains functional requirements, it will show a notification of successful upload.
Exception:	If System fails to verify the functional requirements, it informs User offile discard.

2.5 Use case design

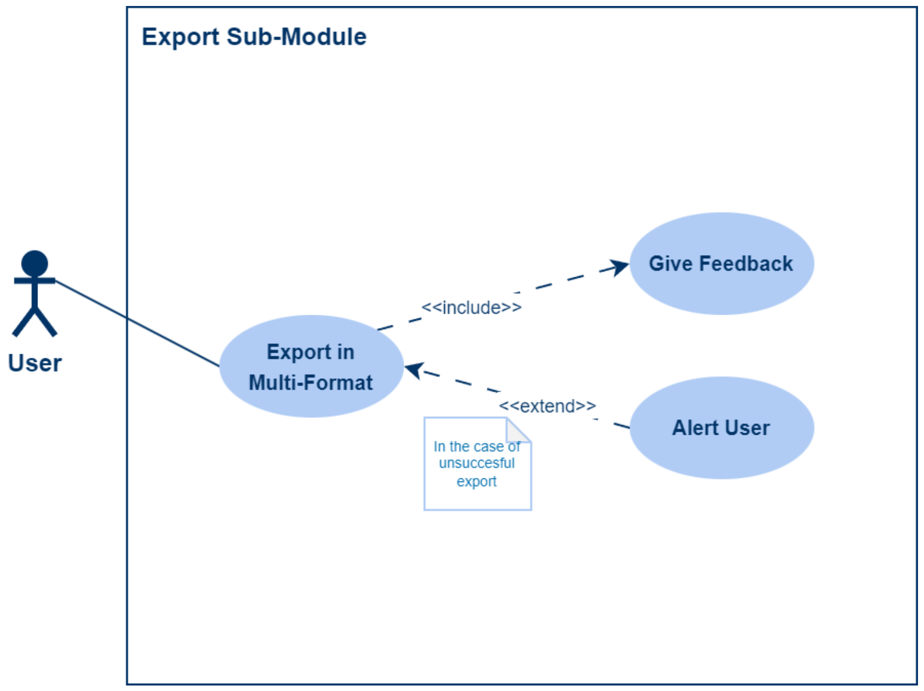


Figure 1: Use Case Diagram of Export Module (UD01)

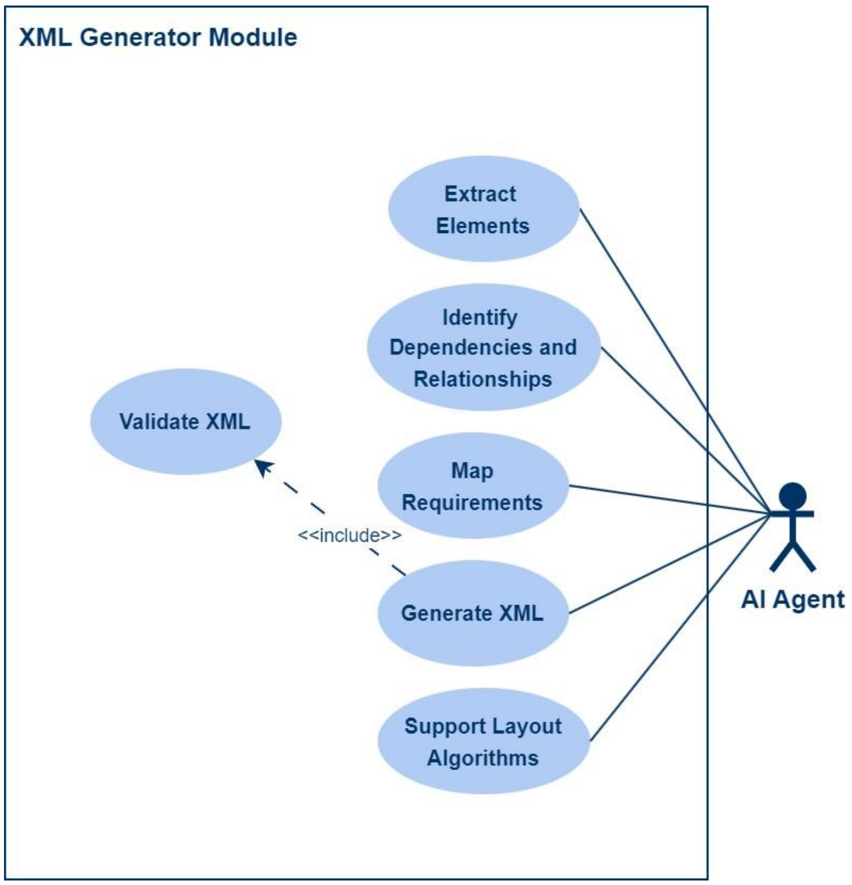


Figure 2: Use Case Diagram of Generator (UD02)

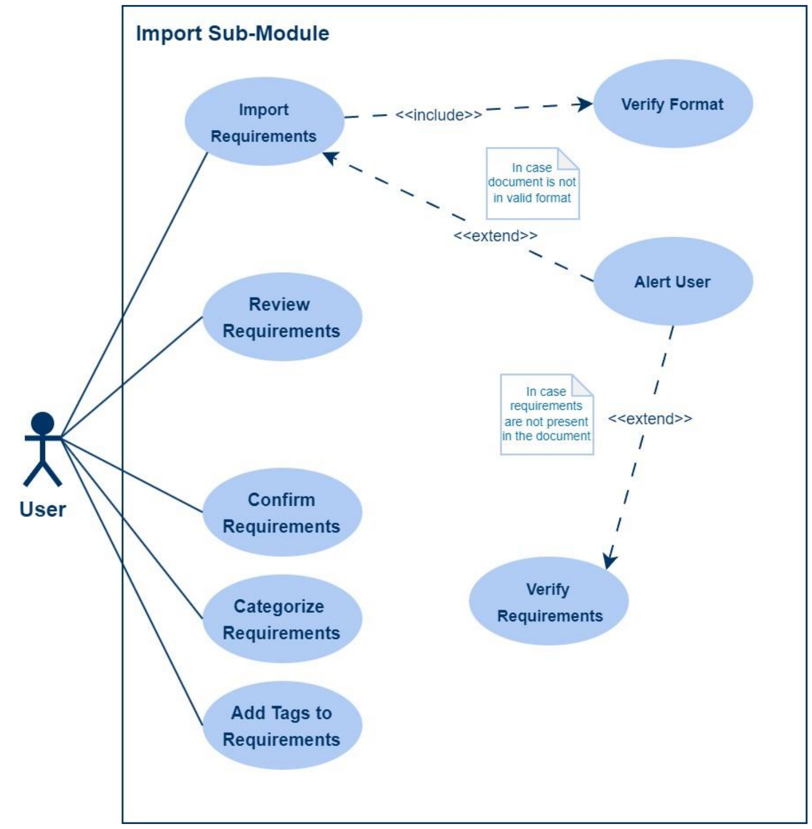


Figure 3: Use Case Diagram of Import (UD03)

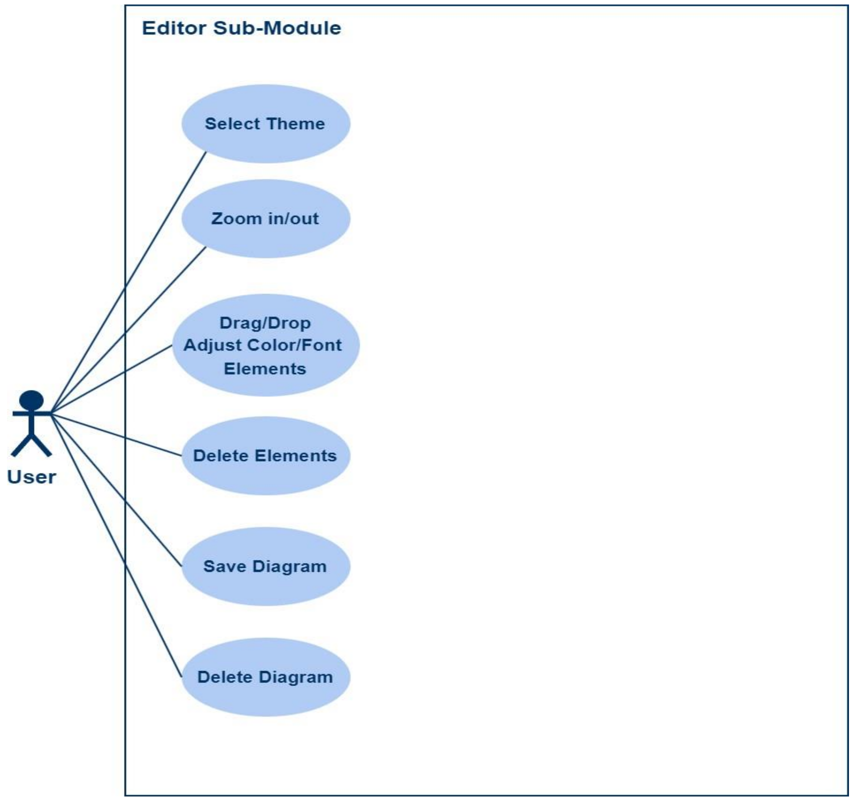


Figure 4: Use Case Diagram of Export(UD04)

2.6 Software development life cycle model

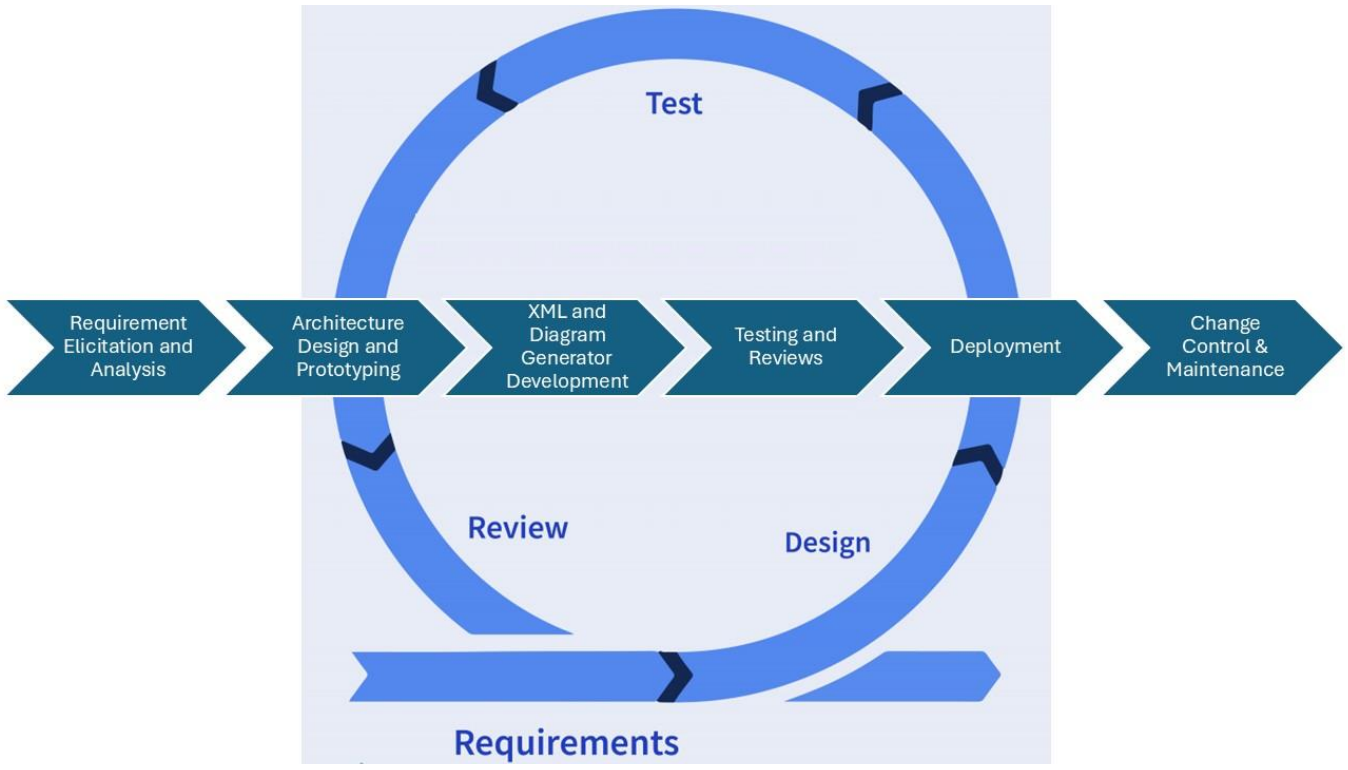
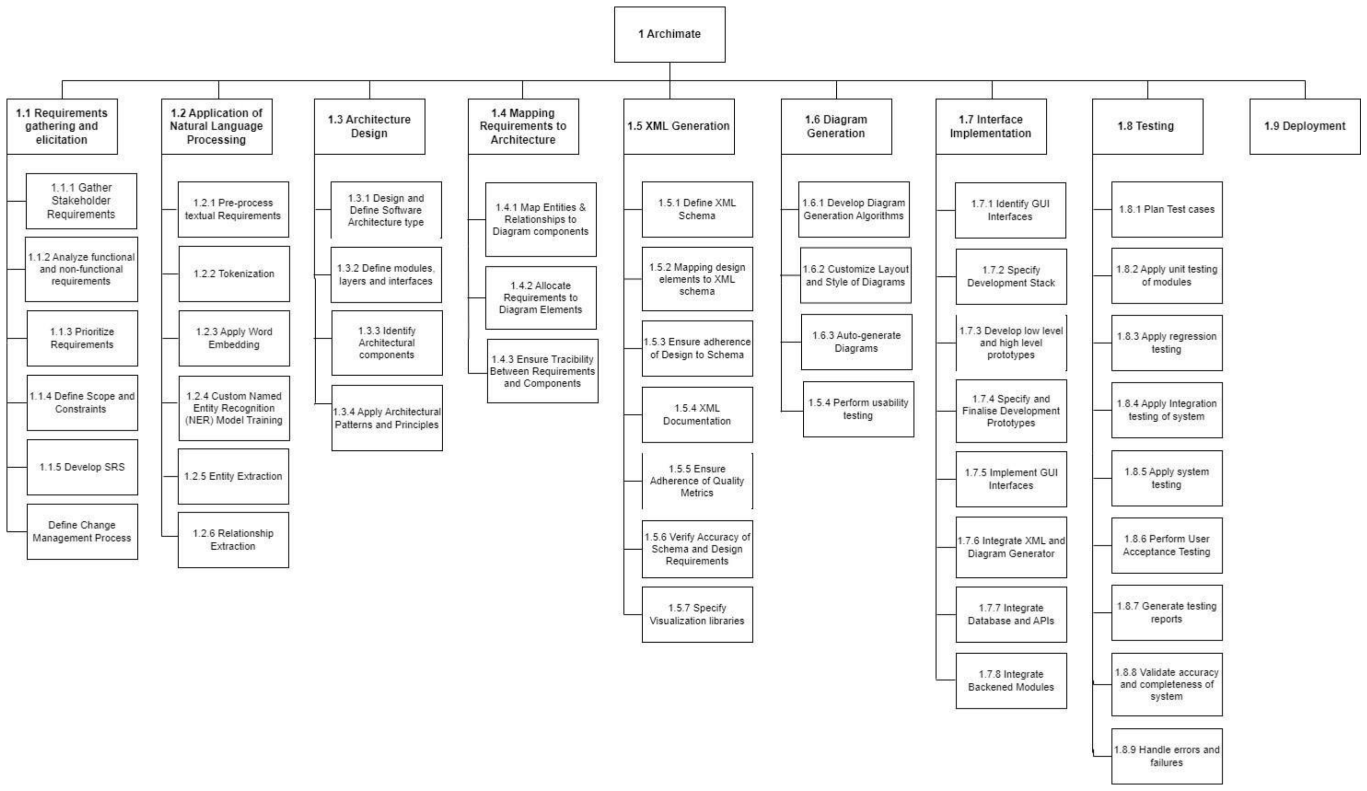

Test
Requirement Elicitation and Analysis
Architecture Design and Prototyping
XML and Diagram Generator Development
Testing and Reviews
Deployment
Change Control & Maintenance
Review
Design
Requirements

Figure 5: Project's Software Development Life Cycle

3 System Design

3.1 Work breakdown structure (WBS)



1 Archimate
1.1 Requirements gathering and elicitation
1.1.1 Gather Stakeholder Requirements
1.1.2 Analyze functional and non-functional requirements
1.1.3 Prioritize Requirements
1.1.4 Define Scope and Constraints
1.1.5 Develop SRS
Define Change Management Process
1.2 Application of Natural Language Processing
1.2.1 Pre-process textual Requirements
1.2.2 Tokenization
1.2.3 Apply Word Embedding
1.2.4 Custom Named Entity Recognition (NER) Model Training
1.2.5 Entity Extraction
1.2.6 Relationship Extraction
1.3 Architecture Design
1.3.1 Design and Define Software Architecture type
1.3.2 Define modules, layers and interfaces
1.3.3 Identify Architectural components
1.3.4 Apply Architectural Patterns and Principles
1.4 Mapping Requirements to Architecture
1.4.1 Map Entities & Relationships to Diagram components
1.4.2 Allocate Requirements to Diagram Elements
1.4.3 Ensure Tracibility Between Requirements and Components
1.5 XML Generation
1.5.1 Define XML Schema
1.5.2 Mapping design elements to XML schema
1.5.3 Ensure adherence of Design to Schema
1.5.4 XML Documentation
1.5.5 Ensure Adherence of Quality Metrics
1.5.6 Verify Accuracy of Schema and Design Requirements
1.5.7 Specify Visualization libraries
1.6 Diagram Generation
1.6.1 Develop Diagram Generation Algorithms
1.6.2 Customize Layout and Style of Diagrams
1.6.3 Auto-generate Diagrams
1.5.4 Perform usability testing
1.7 Interface Implementation
1.7.1 Identify GUI Interfaces
1.7.2 Specify Development Stack
1.7.3 Develop low level and high level prototypes
1.7.4 Specify and Finalise Development Prototypes
1.7.5 Implement GUI Interfaces
1.7.6 Integrate XML and Diagram Generator
1.7.7 Integrate Database and APIs
1.7.8 Integrate Backened Modules
1.8 Testing
1.8.1 Plan Test cases
1.8.2 Apply unit testing of modules
1.8.3 Apply regression testing
1.8.4 Apply Integration testing of system
1.8.5 Apply system testing
1.8.6 Perform User Acceptance Testing
1.8.7 Generate testing reports
1.8.8 Validate accuracy and completeness of system
1.8.9 Handle errors and failures
1.9 Deployment

Figure 6: Work Break Down Structure

3.2 Activity diagram

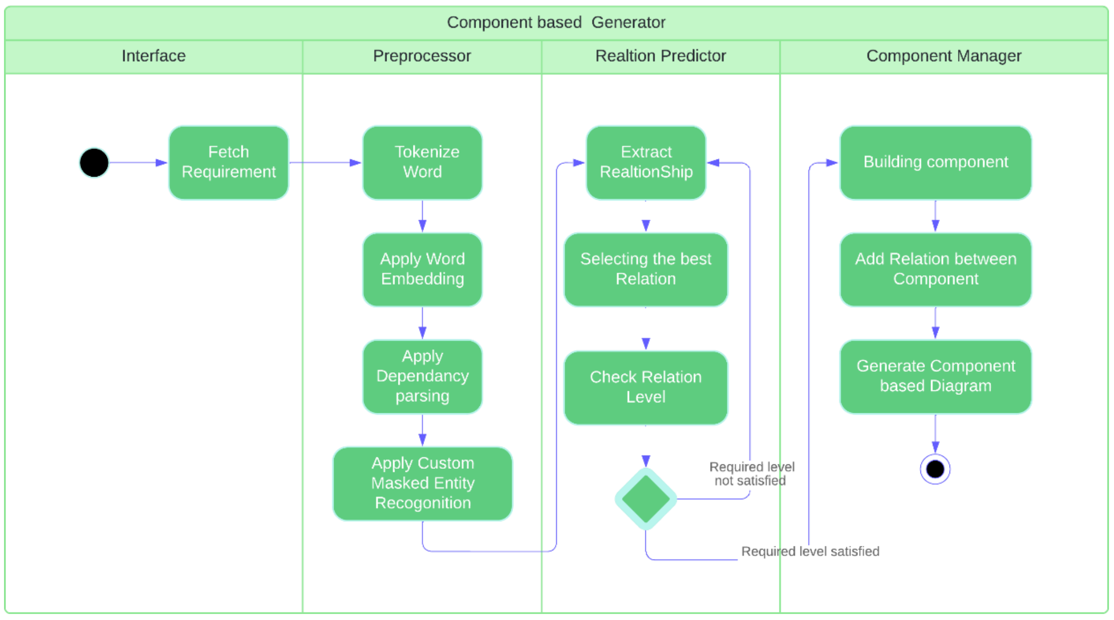


Figure 7: Component Based Generator (AD01)

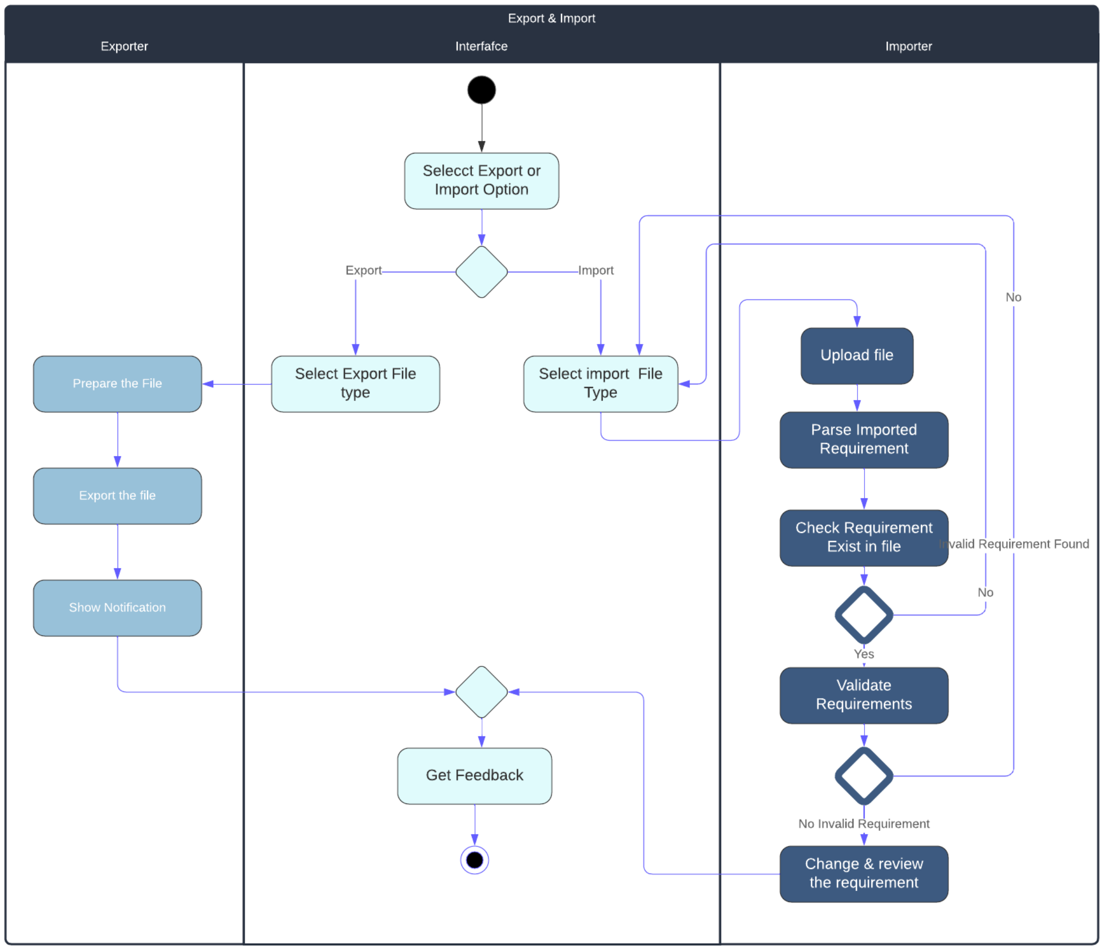


Figure 8: Import & Export (AD02)

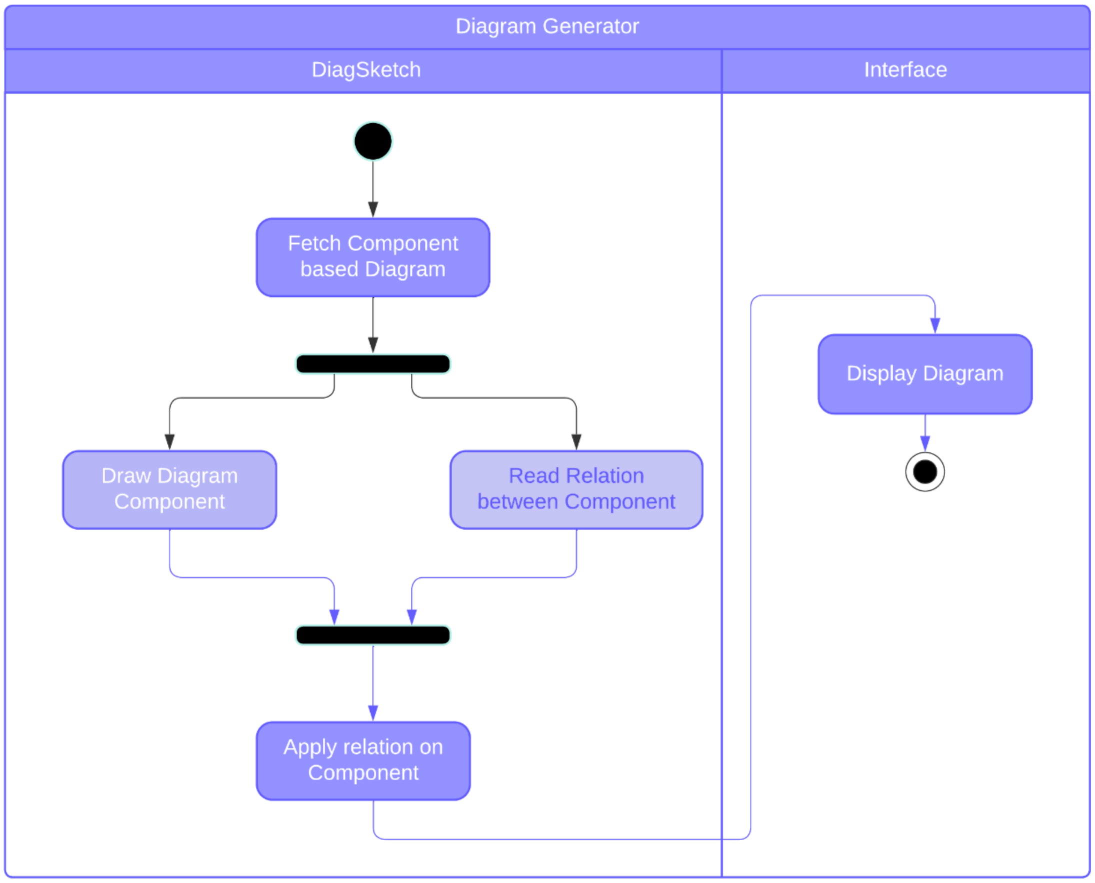


Diagram Generator
DiagSketch
Interface
Fetch Component based Diagram
Draw Diagram Component
Read Relation between Component
Apply relation on Component
Display Diagram

Figure 9: Diagram Generator (AD03)

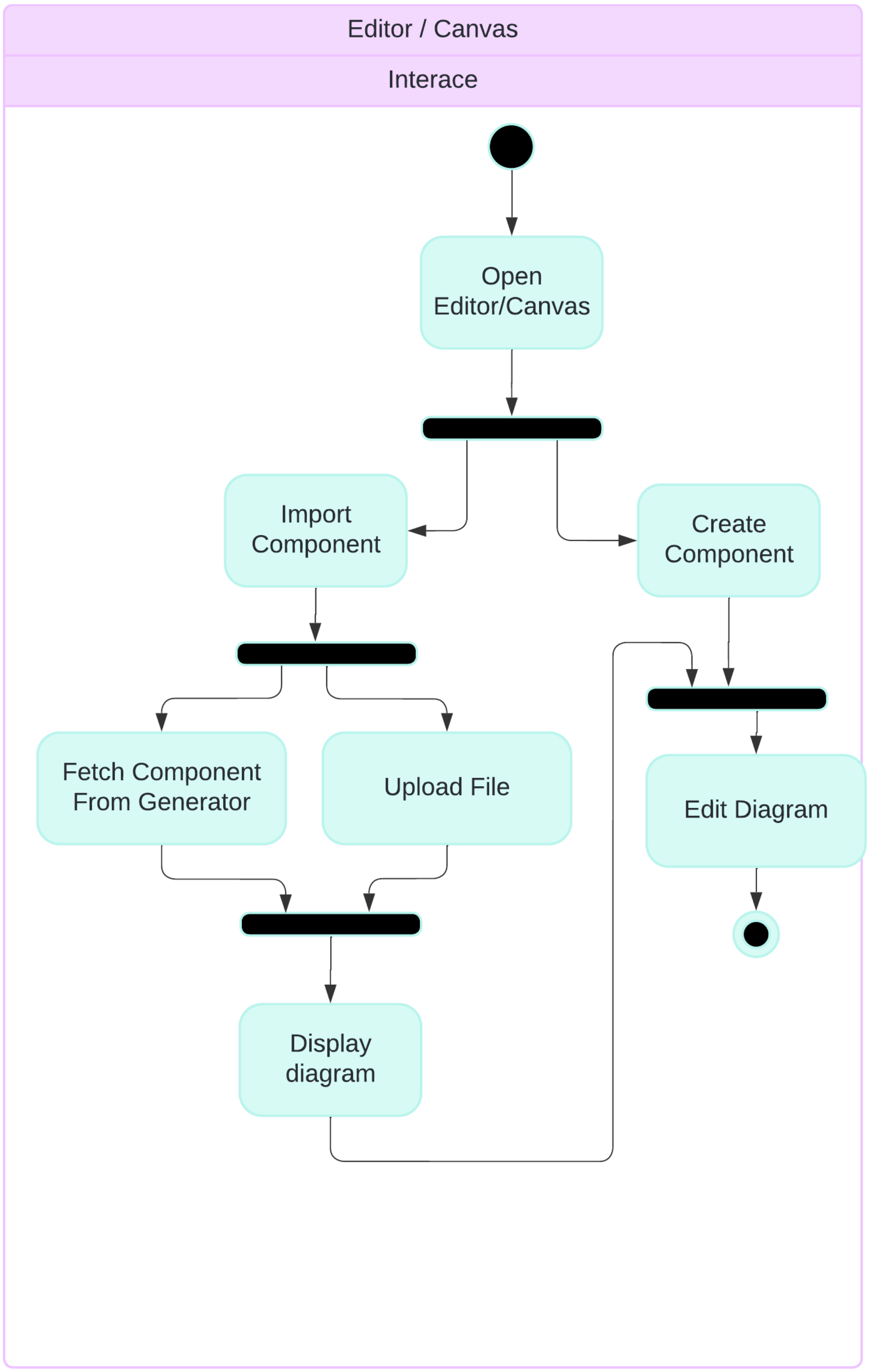


Figure 10: Editor/Canvas (AD04)

3.3 Sequence diagram

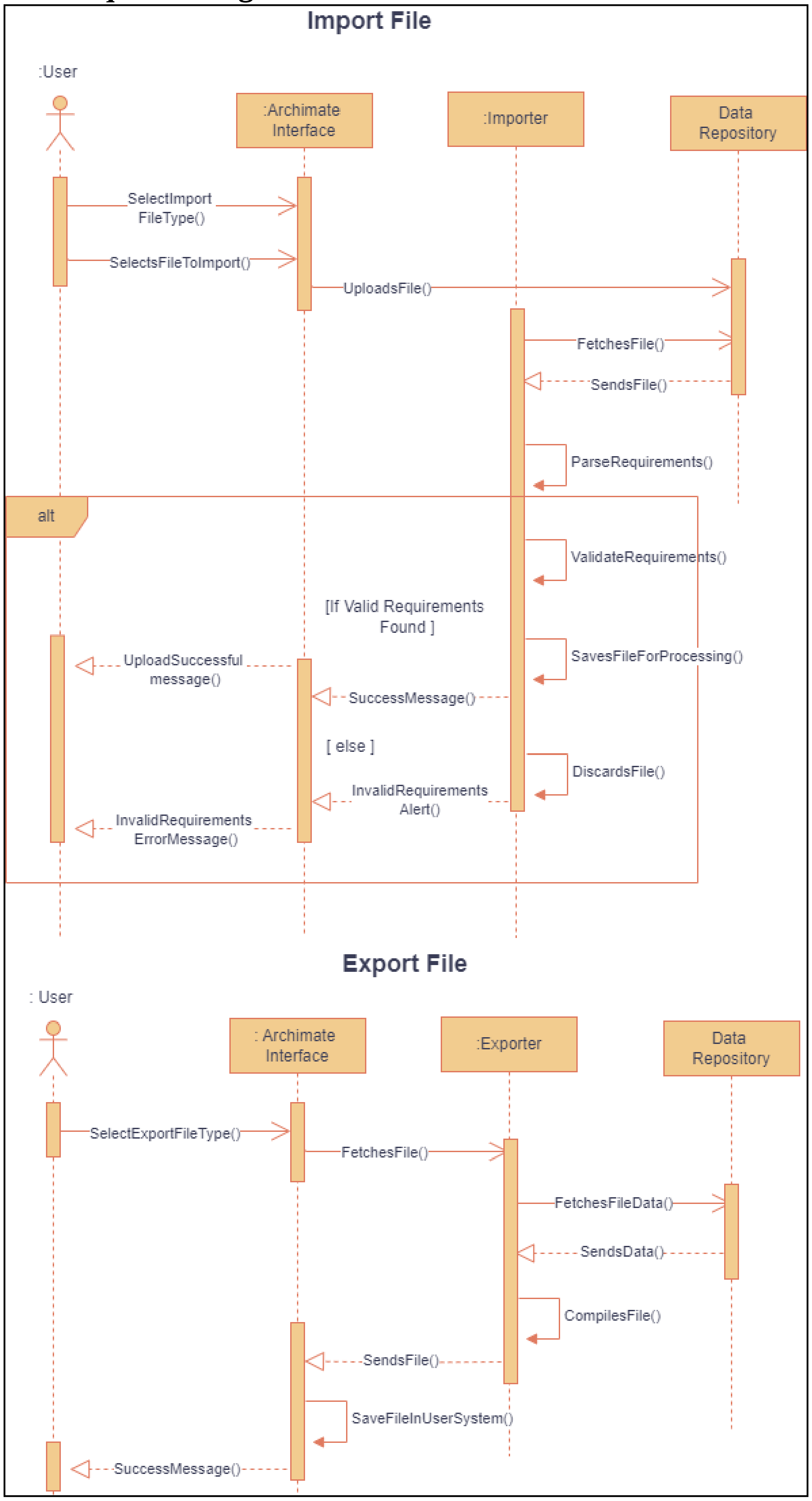


Figure 11: Sequence Diagram of Import/Export (SD01)

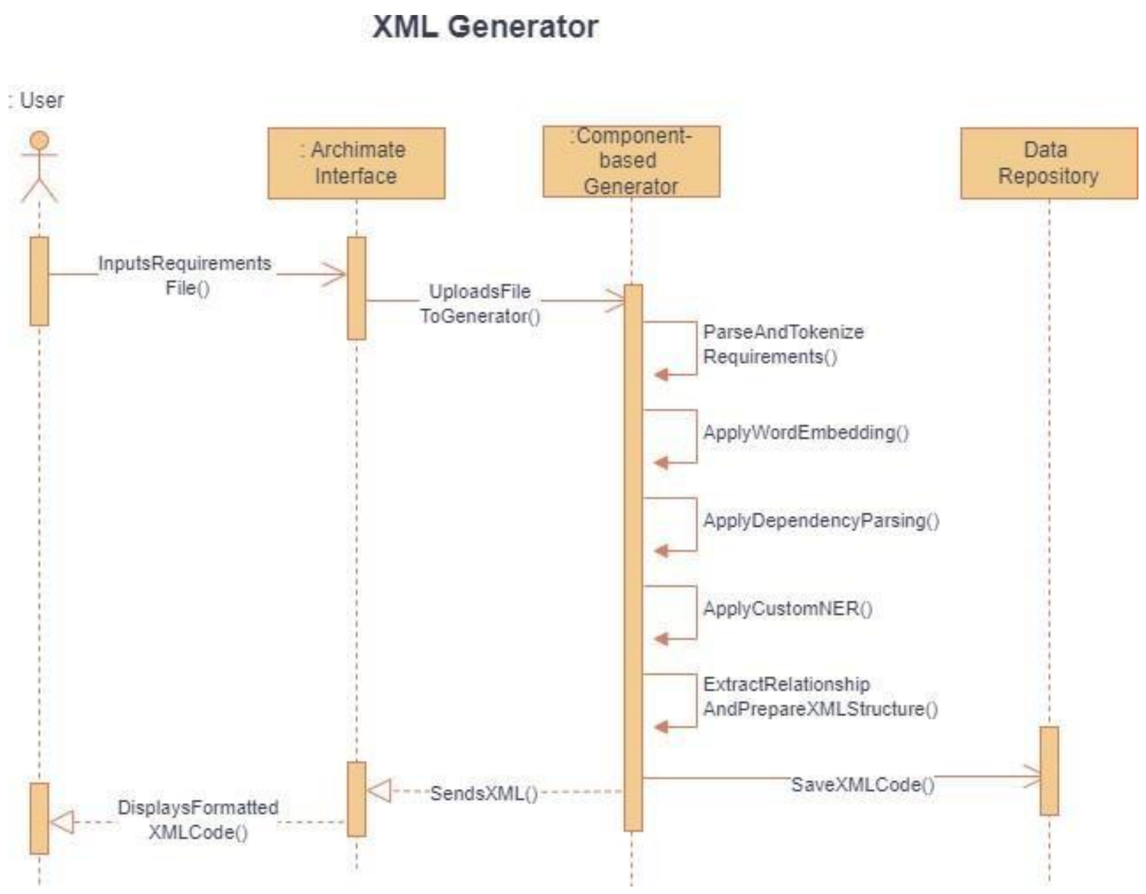


Figure 12: Sequence Diagram of Component Based Generator (SD02)

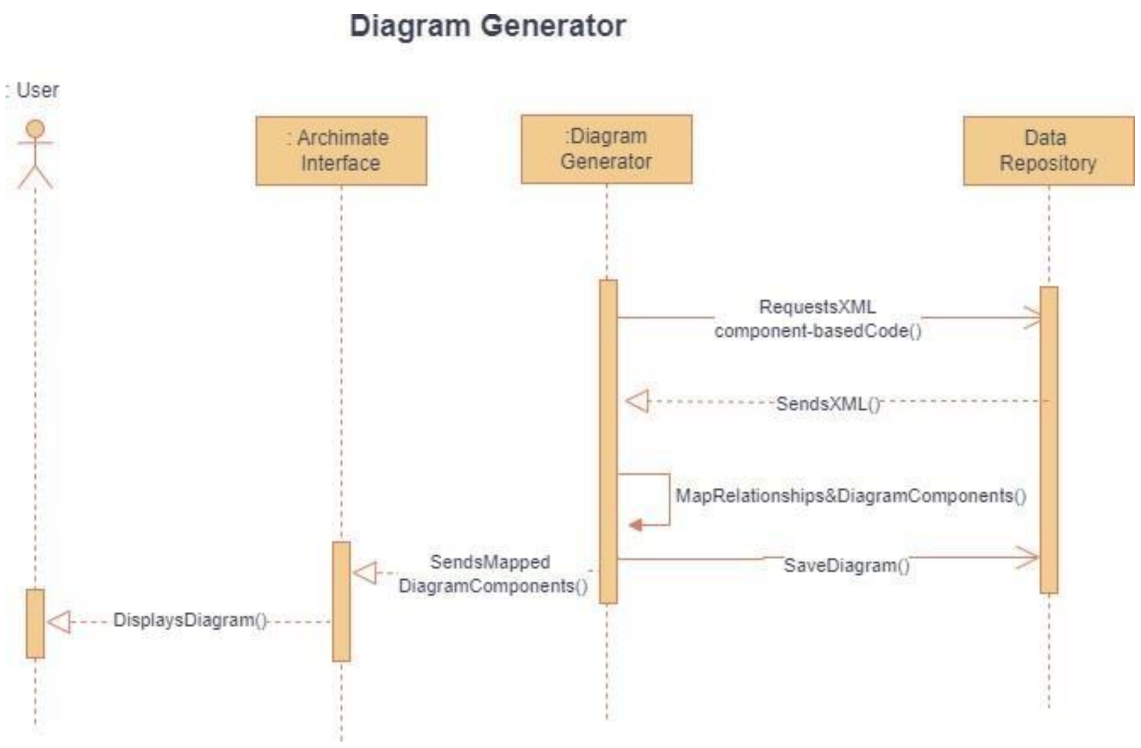


Figure 13: Sequence Diagram of Diagram Generator (SD03)

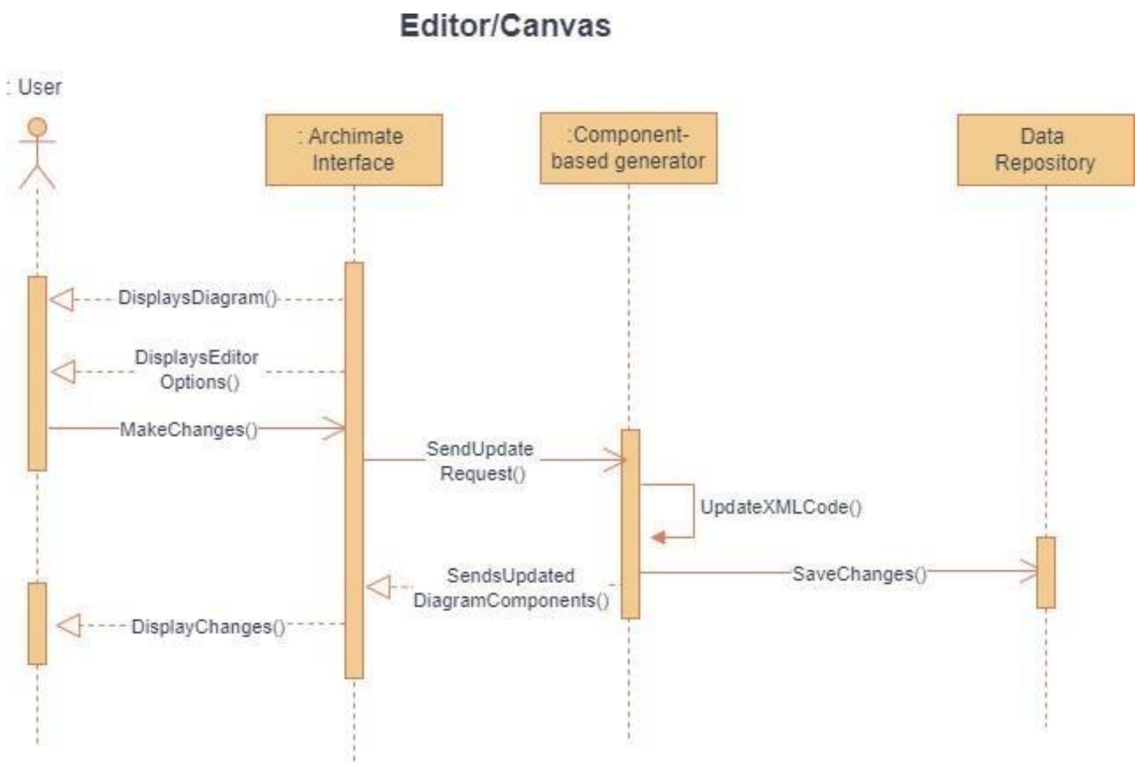


Figure 14: Sequence Diagram of Editor/Canvas (SD04)

3.4 Software architecture

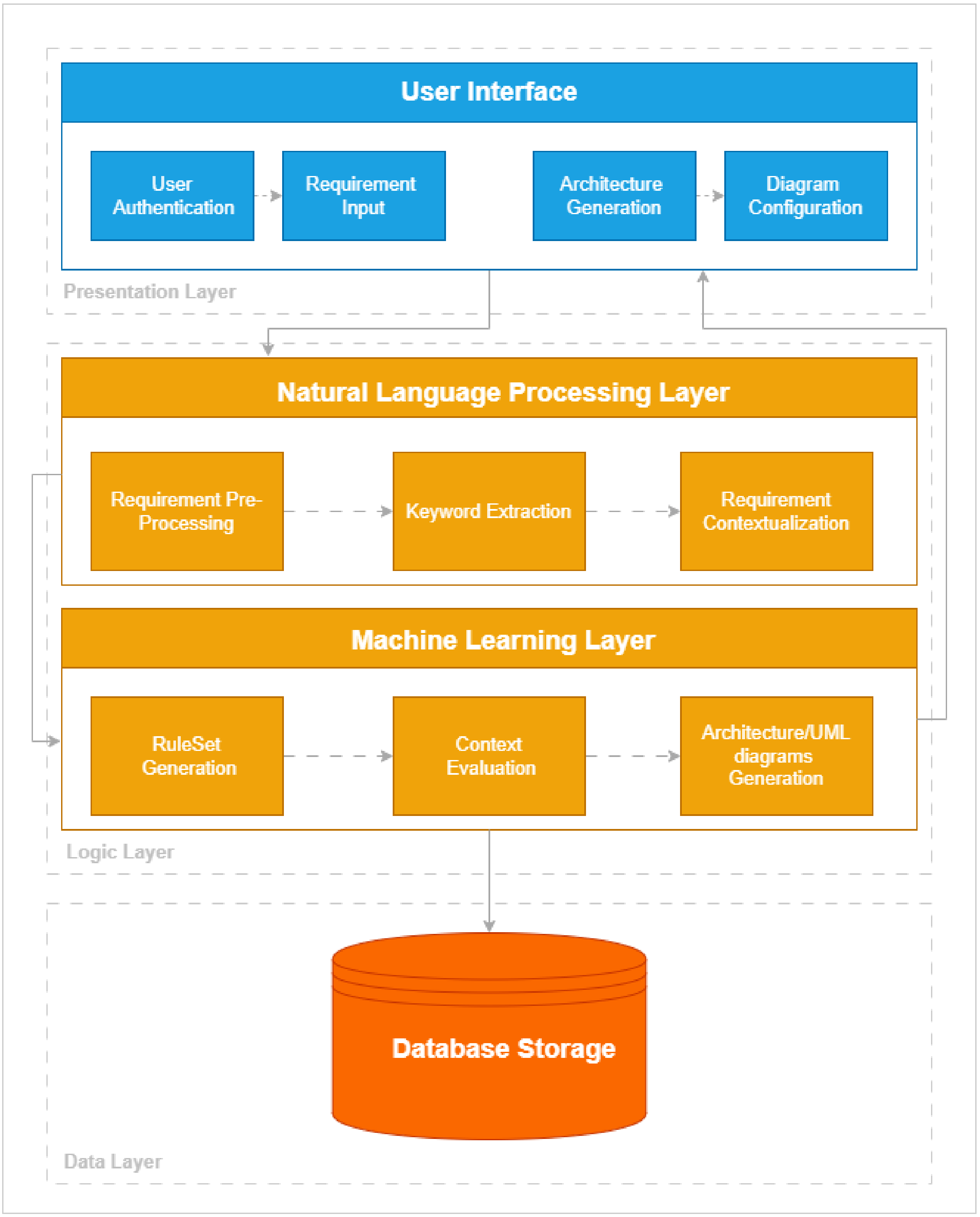


Figure 15: Diagram of System Architecture of RBDG

3.5 Network diagram (Gantt chart)

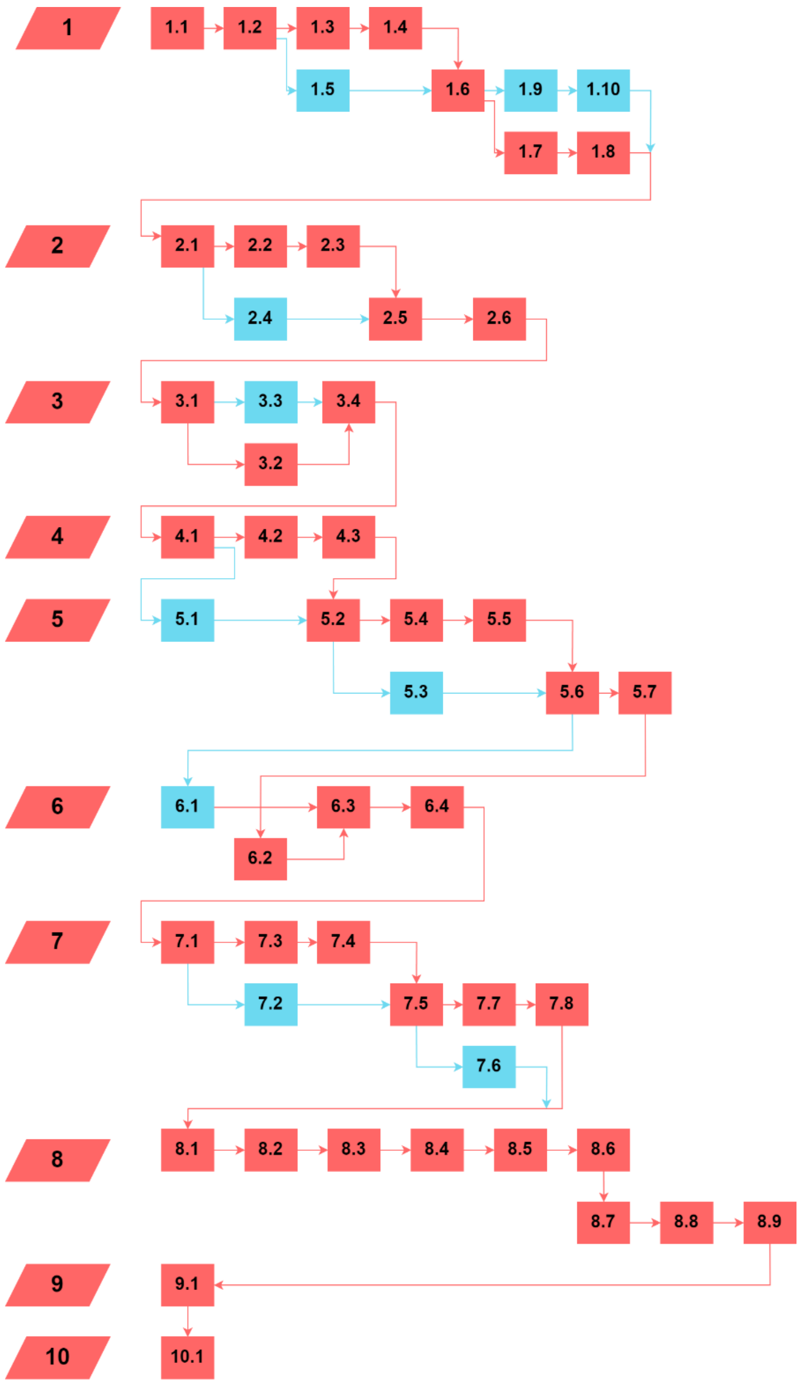
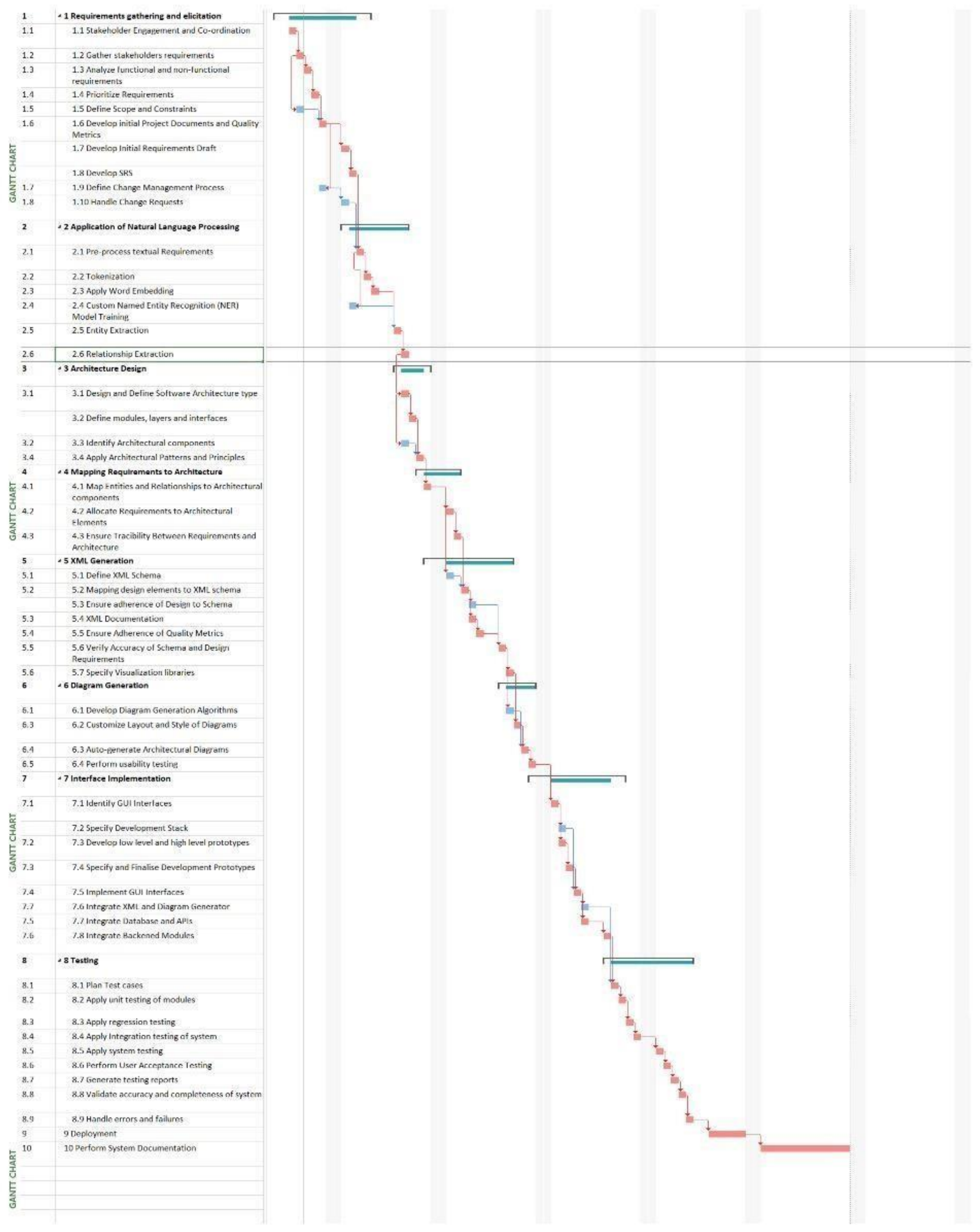


Figure 18: Project Network Diagram



1	**1 Requirements gathering and elicitation**
1.1	1.1 Stakeholder Engagement and Co-ordination
1.2	1.2 Gather stakeholders requirements
1.3	1.3 Analyze functional and non-functional requirements
1.4	1.4 Prioritize Requirements
1.5	1.5 Define Scope and Constraints
1.6	1.6 Develop initial Project Documents and Quality Metrics
	1.7 Develop Initial Requirements Draft
	1.8 Develop SRS
1.7	1.9 Define Change Management Process
1.8	1.10 Handle Change Requests
2	**2 Application of Natural Language Processing**
2.1	2.1 Pre-process textual Requirements
2.2	2.2 Tokenization
2.3	2.3 Apply Word Embedding
2.4	2.4 Custom Named Entity Recognition (NER) Model Training
2.5	2.5 Entity Extraction
2.6	2.6 Relationship Extraction
3	**3 Architecture Design**
3.1	3.1 Design and Define Software Architecture type
	3.2 Define modules, layers and interfaces
3.2	3.3 Identify Architectural components
3.4	3.4 Apply Architectural Patterns and Principles
4	**4 Mapping Requirements to Architecture**
4.1	4.1 Map Entities and Relationships to Architectural components
4.2	4.2 Allocate Requirements to Architectural Elements
4.3	4.3 Ensure Tracibility Between Requirements and Architecture
5	**5 XML Generation**
5.1	5.1 Define XML Schema
5.2	5.2 Mapping design elements to XML schema
	5.3 Ensure adherence of Design to Schema
5.3	5.4 XML Documentation
5.4	5.5 Ensure Adherence of Quality Metrics
5.5	5.6 Verify Accuracy of Schema and Design Requirements
5.6	5.7 Specify Visualization libraries
6	**6 Diagram Generation**
6.1	6.1 Develop Diagram Generation Algorithms
6.3	6.2 Customize Layout and Style of Diagrams
6.4	6.3 Auto-generate Architectural Diagrams
6.5	6.4 Perform usability testing
7	**7 Interface Implementation**
7.1	7.1 Identify GUI Interfaces
	7.2 Specify Development Stack
7.2	7.3 Develop low level and high level prototypes
7.3	7.4 Specify and Finalise Development Prototypes
7.4	7.5 Implement GUI Interfaces
7.7	7.6 Integrate XML and Diagram Generator
7.5	7.7 Integrate Database and APIs
7.6	7.8 Integrate Backened Modules
8	**8 Testing**
8.1	8.1 Plan Test cases
8.2	8.2 Apply unit testing of modules
8.3	8.3 Apply regression testing
8.4	8.4 Apply Integration testing of system
8.5	8.5 Apply system testing
8.6	8.6 Perform User Acceptance Testing
8.7	8.7 Generate testing reports
8.8	8.8 Validate accuracy and completeness of system
8.9	8.9 Handle errors and failures
9	9 Deployment
10	10 Perform System Documentation

Figure19 : Project Gant Chart

4 System Testing

4.1 Test cases

Table 5: Test Case

Test case ID	TC_04	**Test case description**	Test whether System is successfully able to fetch the input requirementsfrom the import module.		
Created by:	Abdullah	**Reviewed by**	Nouman	**Version No.**	1.0
QA Tester's log	Nil.				
Tester's name	Miss Mamoona	**Date testesd:**	Nil	**Test case(Pass/Fail/Not Executed)**	Not Executed

S #	**Prerequisites:**	**S #**	**Test Data**
1	The System has parsed the requirements.	1	Parsed Requirements.
2		2	
3		3	
4		4	

Test Scenario After User has imported the requirements and System has parsed, System fetches the parsed for further processing.

Step No.	**Step details**	**Expected results**	**Actual results**	**Pass /Fail /Not Executed /Suspended**
1	System fetches the imported and parsed requirements.	The requirements are properly fetched.	The requirements areproperly fetched.	Pass

Test case ID	TC_06	**Test case description**	Test whether User is able to input the components to Diagram.		
Created by:	Abdullah	**Reviewed by**	Javeria	**Version No.**	1.0
QA Tester's log	Nil.				
Tester's name	Miss Mamoona	**Date testesd:**	Nil	**Test case(Pass/Fail/Not Executed)**	Not Executed

S #	**Prerequisites:**	**S #**	**Test Data**
1	Diagram has been generated.	1	Generated Diagram
2	Diagram is represented and is editable.	2	
3		3	

Test Scenario	User			
Step No.	**Step details**	**Expected results**	**Actual results**	**Pass /Fail /Not Executed /Suspended**
1	User clicks on the import requirement button.	System allows User to browse throughtheir local storage.	System allows User to browse through their local storage.	Pass
2	User selects thedesired document.	System loads the selected document.	System loads theselected document.	Pass
3	System validates the document by checking the requirements.	System notifies User that the requirements are successfully imported and validated.	System notifies User that the requirements aresuccessfully imported and validated.	Pass
4				

Test case ID	TC_07	**Test case description**	Test the functionality of visualization of diagram.		
Created by:	Nouman	**Reviewed by**	Javeria	**Version No.**	1.0
QA Tester's log	N/A				
Tester's name	Javeria, Abdullah	**Date testesd:**	Nil	**Test case(Pass/Fail/Not Executed)**	Not Executed

S #	**Prerequisites:**		**S #**	**Test Data**
1	Access to Parse Requirements		1	
2			2	

3		3	
4		4	

Test scenario	Verify that System show a graphical representation of Diagram based on requirements.

Step No.	Step details	Expected results	Actual results	Pass /Fail /Not Executed /Suspended
1	Upload the file containing the requirements of System	File should upload.	File should upload.	Pass
2	Confirm the requirements have been successfully uploaded without any mistakes.	Users can review and confirm the requirements.	Users can review and confirm the requirements.	Pass
3	Click on generate.	Diagram displayed to User.	Diagram displayed to User.	Pass

Test case ID	TC_08	**Test case description**	Test the creation of diagram		
Created by:	Javeria	**Reviewed by**	Nouman,Abdullah	**Version No.**	1.0
QA Tester's log					
Tester's name	Nouman,Abdullah	**Date testesd:**	Nil	**Test case(Pass/Fail/Not Executed)**	Not Executed

S #	Prerequisites:
1	Canvas is displayed
2	
3	
4	

S #	Test Data
1	
2	
3	
4	

Test scenario	User should be able to edit/custoize Diagrams that system generated from user requirements

Step No.	Step details	Expected results	Actual results	Pass /Fail /Not Executed /Suspended
1	Component are created	Component are visible	Component are visible	Pass
9	user saves diagram	Diagram should be saved	Diagram should be saved	Pass

Test case ID	TC_09	**Test case description**	Test the customization/editing of diagram		
Created by:	Javeria	**Reviewed by**	Nouman,Abdullah	**Version No.**	1.0

QA Tester's log

Tester's name	Nouman,Abdullah	**Date testesd:**	Nil	**Test case(Pass/Fail/Not Executed)**	Not Executed

S #	Prerequisites:		S #	Test Data
1	Diagram is generted and dislayed		1	
2	Customization options are available and visible.		2	

Test scenario	User should be able to create Diagram from scratch

Step No.	Step details	Expected results	Actual results	Pass /Fail /Not Executed /Suspended
1	User links the component with other component	Component can be linked	Component can belinked	Pass
2	user moves a component to different position	component can be moved	component can bemoved	Pass

Test case ID	TC_10	**Test case description**	Test the saving and deletion of diagram		
Created by:	Javeria	**Reviewed by**	Nouman,Abdullah	**Version No.**	1.0
QA Tester's log					
Tester's name	Nouman,Abdullah	**Date testesd:**	Nil	**Test case(Pass/Fail/Not Executed)**	Not Executed

S #	Prerequisites:		S #	Test Data
1	Diagram is generted and dislayed		1	
2	Saving options are available and visible.		2	

Test scenario	User should be able to save and delete Diagrams that system generated from user requirements

Step No.	Step details	Expected results	Actual results	Pass /Fail /Not Executed /Suspended
1	user selects saving format type	formats should be avalaible and can be selected	formats should be avalaible and can be selected	Pass
2	user clicks on save/download option	option should be selected and file should be downlloaded/saved	option should beselected and file should be downlloaded/saved	Pass
3	user selects Diagram	diagram file should be selected	diagram file should beselected	Pass
4	user deletes the file from archimate	diagram should be deleted from archimate and diagram deleted notification arrives	diagram should be deleted from archimateand diagram deleted notification arrives	Pass

Test case ID	TC_11	**Test case description**	Test the functionality of importing user requirements		
Created by:	Javeria	**Reviewed by**	Nouman,Abdullah	**Version No.**	1.0

QA Tester's log

Tester's name	Nouman,Abdullah	**Date testesd:**	Nil	**Test case(Pass/Fail/Not Executed)**	Not Executed

S #	**Prerequisites:**
1	
2	
3	
4	

S #	**Test Data**
1	File with user requirements
2	
3	
4	

Test scenario	User should import user requirements file

Step No.	**Step details**	**Expected results**	**Actual results**	**Pass /Fail /Not Executed /Suspended**
1	User selects the import file type	file type should be selected	file type should beselected	Pass
2	User clicks the upload button	file should be uploaded and contents are displayed	file should be uploaded and contents aredisplayed	Pass

Test case ID	TC_12	**Test case description**	Test the functionality of parsing User requirements

Created by:	Javeria	**Reviewed by**	Nouman,Abdullah	**Version No.**	1.0

QA Tester's log

Tester's name	Nouman,Abdullah	**Date testesd:**	Nil	**Test case(Pass/Fail/Not Executed)**	Not Executed

S #	**Prerequisites:**
1	Requirements file is uploaded
2	
3	
4	

S #	**Test Data**
1	file with functional requirements
2	file without functional requirements
3	
4	

Test scenario	System should parse the input user requirements

Step No.	**Step details**	**Expected results**	**Actual results**	**Pass /Fail /Not Executed /Suspended**
1	User uploads the file	file should be uploaded and parsed	file should be uploadedand parsed	PAss

Test case ID	TC_13	**Test case description**	Test the functionality of validation of requirements.		
Created by:	Nouman	**Reviewed by**	Abdullah	**Version No.**	1.0

QA Tester's log	Nil

Tester's name	Javeria, Abdullah	**Date testesd:**	Nil	**Test case(Pass/Fail/Not Executed)**	Not Executed

S #	**Prerequisites:**
1	
2	
3	
4	

S #	**Test Data**
1	
2	
3	
4	

Test scenario	Verify that System validates the requirement uploaded by User.

Step No.	**Step details**	**Expected results**	**Actual results**	**Pass /Fail /Not Executed /Suspended**
1	Upload the file containing the requirements of System	The file should upload.	The file should upload.	Pass
2	Click on the upload.	Requirements assigned in a variable	Requirements assignedin a variable	Pass

Test case ID	TC_14	**Test case description**	Test the functionality of Confirmation of requirements from User.

Created by:	Nouman	**Reviewed by**	Javeria	**Version No.**	1.0

QA Tester's log	Nil

Tester's name	Javeria, Abdullah	**Date testesd:**	Nil	**Test case(Pass/Fail/Not Executed)**	Not Executed

S #	**Prerequisites:**	**S #**	**Test Data**
1	User already uploaded the requirement file.	1	
2	Requirements have been parsed in a variable.	2	
3		3	
4		4	

Test scenario	Verify that System validates the requirement uploaded by User.

Step No.	**Step details**	**Expected results**	**Actual results**	**Pass /Fail /Not Executed /Suspended**
1	Review the Requirement to find any mistake	Requirements should be displayed on the interface for review.	Requirements should bedisplayed on the interface for review.	Pass
2	Change the mistake if any.	Changes made in the requirements.	Changes made in therequirements.	Pass
3	Click on confirm.	Requirement status changed to confirm	Requirement statuschanged to confirm	Pass

Test case ID	TC_15	**Test case description**	Test the functionality of Customization Options on requirements.		
Created by:	Nouman	**Reviewed by**	Abdullah	**Version No.**	1.0
QA Tester's log	Nil				
Tester's name	Javeria, Abdullah	**Date testesd:**	Nil	**Test case(Pass/Fail/Not Executed)**	Not Executed

S #	**Prerequisites:**	**S #**	**Test Data**
1	User already uploaded the requirement file.	1	
2	Requirements have been parsed in a variable.	2	
3	Requirements have been Reviewed by User.	3	
4		4	

Test scenario	Verify that System validates the requirement uploaded by User.

Step No.	**Step details**	**Expected results**	**Actual results**	**Pass /Fail /Not Executed /Suspended**
1	Click on add meta data.	Selection page for metadata should display.	Selection page for metadata is display.	Pass
2	Enter metadata/annotation of requirement	Data enter in the fields.	Data enter in the fields.	Pass

3	Click on submit	Metadata added in requirement detail	Metadata added inrequirement detail	Pass

Test case ID	TC_16	**Test case description**	Test the functionality of Exporting Diagram		
Created by:	Nouman	**Reviewed by**	Javeria	**Version No.**	1.0
QA Tester's log	Nil				
Tester's name	Javeria, Abdullah	**Date testesd:**	Nil	**Test case(Pass/Fail/Not Executed)**	Not Executed

S #	**Prerequisites:**	**S #**	**Test Data**
1	The requirement file is uploaded.	1	
2	Requirements have been reviewed.	2	
3	Diagram of requirement has been generated.	3	
4		4	

Test scenario Verify that System validates the requirement uploaded by User.

Step No.	**Step details**	**Expected results**	**Actual results**	**Pass /Fail /Not Executed /Suspended**
1	Click on File, Then Export	Export option got selected from drop down menu.	Export option got selected from drop down menu.	Pass

2	Select the file type of Visio (.vsdx)	Visio file type selected, and Json file exported successfully.	Visio file type selected,and Visio file exported successfully.	Pass

Test case ID	TC_17	**Test case description**	Test the functionality of feedback.		
Created by:	Nouman	**Reviewed by**	Javeria	**Version No.**	1.0
QA Tester's log	N/A				
Tester's name	Javeria, Abdullah	**Date testesd:**	N/A	**Test case(Pass/Fail/Not Executed)**	Not Executed

S #	**Prerequisites:**	**S #**	**Test Data**
1	Requirement file is uploaded.	1	
2	Requirements have been reviewed.	2	
3	Diagram of requirement has been generated.	3	
4	File Exported Successfully	4	

Test scenario	Verify that system validates the requirement uploaded by User.

Step No.	**Step details**	**Expected results**	**Actual results**	**Pass /Fail /Not Executed /Suspended**
1	Check the file exported Successfully notifications	The file has been exported successfully.	The file has been exported successfully.	Pass

4.2 Unit / integration / acceptance testing

4.2.1 Unit testing:

Unit testing is one of the fundamental parts of the software testing phase where individual components are tested in isolation. The primary goal of this testing is to validate the functionality of each unit and whether they perform as expected. For the unit testing, here are some techniques that will be performed.

Black Box Testing:

This technique of testing involves testing individual components but with the condition that we do not dive deep into the internal structure or the inner workings of the components. This will be most useful when verifying whether the individual components meet the specified requirements or not.

White Box Testing:

Glass box or transparent testing involves testing the unit while also peering in the internal structure and the workings of the unit. Here, the tester will have full visibility of the code including paths, loop conditions, etc. This approach is a must considering we want to have efficiency and ensure no unwanted behaviour is occurring.

Parameterized Testing:

This technique will involve running the same test cases with multiple sets of input parameters hence validating the unit’s functionality over a wide range of inputs and ensuring that all cases are dealt with.

4.2.2 Integration testing:

It is an important part of testing where individual units of software are combined together and then tested as a group. The main aim is to ensure that integrated components work together as expected. Here are some techniques to be used in this project for integration testing:

Top-Down Testing:

It will test the top-level modules first and then move down to lower level modules by gradually integrating them. Stubs (dummy implementation of lower-level modules) are used to simulate the behavior of lower level modules until they are integrated. This technique helps in identifying the and resolving major issues in the early phase of development.

Bottom-Up Testing:

This is the opposite of top-down approach. It begins by testing the lower level modules first and then gradually moves to integrating and testing the higher level modules. Drivers (dummy implementation of higher level modules) are used to simulate the behavior of higher level modules until they are integrated. This approach will be easier to test the lower level modules.

4.2.3 Acceptance testing:

Alpha testing:

Alpha testing will be conducted by the internal team or a selected group of users in a controlled environment. It aims to gathers defects in the early stage of development process and gather feedback for improvements before releasing it to the targeted users.

Beta-testing:

In this testing, the software will be tested by allowing its access to a limited no of external users to gather feedback on its performance, usability and other evaluation metrics. This will help in identifying the issues that was not discovered during the early phases of testing.

5 Conclusion

5.1 Problems faced and lessons learned

5.1.1 Problem Faced

1. The first major problem we encountered, concerned about the availability of a labeled dataset that could be reused directly in our model, especially in terms of software architecture. This misalignment became a problem when it affected feature engineering, one of the essential phases in a machine learning pipeline. Overcoming this required understanding the architecture of the system and the nature of the data to create a structure, then entering the images data manually into the structure and performing feature extraction and general alignment with the model needs.
2. One more important problem we experienced was that the stakeholder was missing, and the person, who initially described the requirements, was also missing. This lack of direct input led to several difficulties especially when assessing the software architecture. Most traditional approaches to architectural evaluation including SASEM, and Reliability Analysis involve stakeholders in the definition of, and elaboration of the evaluation criteria. In the absence of such inputs we had to set our own standards and benchmarks in respect of the architectural layout.
3. One of the most used evaluation techniques, scenario-based, which is used to determine the ability of an architecture to achieve certain scenarios and quality attributes, depends on stakeholders to develop scenarios. These scenarios, which are used to describe interactions between some stakeholders and the system, make the thought process of the architectural decisions clearer.
4. Yet another problem, which we could encounter, was lack of the stakeholder, as the person who stated the requirements eliminates from the process. Such absence of direct input gave rise to several concerns and challenges, and most notably the assessment of the software architecture. A major shortcoming of most well-accepted architectural

evaluation processes like (ed.).SSEAME & RA are the facts that these processes need the identification and elaboration of assessment criteria with stakeholders.

5. Further, most of the techniques do not take into account that requirements may change as the architecture is evaluated, sequential refinements are possible. This assumption was not possible in our case, we had a list of requirements that remained constant regardless of what the stakeholders had to say.
6. Indeed, most of the conventional approaches for evaluation and selection were not suitable or optimal in our case. Furthermore, there is little information about the methods on how the architectures produced from Machine Learning methods can be assessed field, which makes evaluation even more challenging in our case.

5.1.2 Lesson Learned

After facing these challenges, we have learned many lessons. We now understand not rushing the training of the model prematurely. Instead, we need to establish clear goals find proper information and then a workflow and strategy before tackling the task.

We learned the significance of planning comprehensively and focused more on the modules as they break down the complex structure into sizable and understandable chunks. Similarly, this approach would have enhanced the collaboration among us. Another benefit from this was that we got to get hands-on experience with iterative development and how refinement and improvement are carried on.

5.1.3 Project Summary:

We dive into the implementation of our Final Year Project (FYP) hence our main focus remains on automating the generation of architecture and UML diagrams through the functional requirements provided by the user. This is most beneficial in translating requirements from users and getting actual visual representations of them.

We have harnessed the power that resides in advanced Natural Learning Processing (NLP) along with the various algorithms in standard Machine Learning and Deep Learning. By automating the process, we have aimed to enhance the efficiency and productivity for software engineers, and system architects and provide ease to all stakeholders.

Plans have been set forth in dealing with the structure of various system diagrams such as activity diagrams, and use case diagrams, and the most important of them all, the one that provides the sight over the entire system is the architecture of the system. We aim that the final results have flexibility and can be scaled accordingly to what the situation desires.

We have implemented various forms of testing that we have come up with as our focus is not only generating the diagrams but also ensuring that the diagrams are reliable and actually represent/reflect the requirements provided by the user.

To summarize it all, the FYP represents a significant contribution to the problem discussed. Our goal is to ensure that our project ends up actually helping the development process, especially in the domain and art of requirements analysis and diagram creation. Although we expect that things will go our way so easily and we will have to face a mountain of challenges with these challenges, we also find opportunities.

5.1.4 Future Work:

A. **Proposed Usage of High Quality** Dataset Ideally, the core of a big-sized and qualitative dataset should be developed in cooperation with professionals who are more or less involved into the creation of multimedia products. This guarantees the information from the research is total and depicts real existence conditions in the course of the simulations. Of tremendous importance in Machine Learning is the quality of the dataset that has to be fed to models for decision-making.

B. **Training the Model beyond the Diagram Based Generation** This is why the model should not only depict diagrams but should be able to write code or solve a problem. The techniques such as supervised learning and reinforcement learning together with the fine-tuning make the model more appropriate and suitable for the various real-world applications.

6 References

[1] A. S. A. C. Shaohong Zhong, " Natural LanguageProcessingforsystemsengineering:Automatic," *Knowledge-BasedSystems,* vol. 259, no. 0950-7051, p. 110071, 2023.

[2] D. K. a. M. A. B. Deeptimahanti, "An automated tool for generating UML models from natural language requirements.," in *IEEE/ACM International Conference on Automated Software Engineering,* 2009.

[3] P. a. R. P. More, "Generating UML diagrams from natural language specifications," *International Journal of Applied Information Systems (IJAIS),* vol. 1, no. 8, pp. 19-23, 2012.

[4] S. G. a. K. K. B. Richa Sharma, "Automated Generation of Activity and Sequence Diagrams from," in *9th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE),* 2014.

[5] M. Y. L. Gamage, "Automated Software Architecture Diagram Generator using," A Dissertation by Mr Yasitha Lalanga Gamage Submitted in partial fulfillment of the requirements for the BSc (Hons) in Computer Science degree at the University of Westminste, 2023.

Requirements based diagram generator

ORIGINALITY REPORT

17%	9%	2%	17%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Higher Education Commission Pakistan Student Paper	6%
2	Submitted to CSU, San Jose State University Student Paper	2%
3	Submitted to University of Bedfordshire Student Paper	1%
4	www.coursehero.com Internet Source	1%
5	Submitted to University of Westminster Student Paper	<1%
6	Submitted to Northern Regional College - CN-165634 Student Paper	<1%
7	Submitted to National College of Ireland Student Paper	<1%
8	Submitted to University of Canberra Student Paper	<1%
9	www.researchgate.net	

0% detected as AI

The percentage indicates the combined amount of likely AI-generated text as well as likely AI-generated text that was also likely AI-paraphrased.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Detection Groups

1 AI-generated only 0%
Likely AI-generated text from a large-language model.

2 AI-generated text that was AI-paraphrased 0%
Likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.