Software Proposal Document For Project Automated Checking and Grading Tool for Documentation

George Ayman, Nader Amir, Nouran Mohamed, Zeina Hesham
Supervised by: Dr. Salwa Osama, Eng. Nada Ayman

January 18, 2025

Proposal Version	Date	Reason for Change
1.0	23-September-2024	Proposal First version's specifications are defined

Table 1: Document version history

GitHub: Automated Checking and Grading Tool for Technical Documentation

Abstract

The Automated Checking and Grading Tool is a Large Language model (LLM)-based system designed to assist university professors in evaluating Software Requirements Specification (SRS) and Software Design Description (SDD) documents. By leveraging a dataset of previously graded documents, the tool aims to provide consistent, objective assessments while significantly reducing manual effort. Using natural language processing (NLP) and Large Language Models (LLMs), the tool analyzes key aspects of documentation, such as clarity, completeness, adherence to business requirements, and proper use of design diagrams. Additionally, the system integrates a self-learning mechanism that continually improves its evaluation accuracy by incorporating feedback from professors. The final score combines the results of multiple evaluation criteria, delivering a detailed and fair assessment of each document's quality. By automating this process, the tool improves grading consistency and reduces the workload for professors, allowing them to focus on higher-level tasks.

1 Introduction

1.1 Background

Since the early days of software engineering education in the 1970s, clear and well-structured documentation, such as Software Requirements Specification (SRS) and Software Design Description (SDD), has been essential for defining system functionality and design. However, the manual review and grading of these documents have consistently posed challenges, requiring extensive time and effort while introducing variability depending on the examiner's interpretation. To address these long-standing issues, this project proposes the development of an Automated Checking and Grading Tool that evaluates SRS and SDD papers using Large Language Models (LLMs) trained on historical grading data. Similar to automated grading systems introduced in other academic fields, this technology will provide standardized assessments of documentation by evaluating key elements such as completeness, clarity, adherence to business requirements, and proper use of design diagrams. Over time, the grading accuracy improves through a self-learning mechanism that adapts as it processes new data. By reducing manual grading effort and providing fair, objective assessments, the tool will streamline the evaluation process, benefiting both instructors and students by delivering faster and more consistent feedback.

1.2 Motivation

1.2.1 Academic

The need for automated grading tools in academia, particularly for technical documents, has been growing alongside advancements in technology and increases in student enrollment. Historically, the grading of Software Requirements Specifications (SRS) and Software Design Descriptions (SDD) has been a labor-intensive task, requiring professors to manually evaluate documents based on various criteria such as clarity, completeness, and adherence to business requirements. As early as the 1990s, efforts were made to automate grading in simpler fields, such as multiple-choice questions and essays. However, the complexity of technical documents in fields like software engineering posed challenges for automation due to the specialized knowledge required to evaluate both the content and structure of these documents. This problem typically arises during the assessment phase of technical documentation courses, especially in larger classes where individualized attention is difficult to provide. While some attempts have been made to standardize grading rubrics, variability in instructor interpretations persists.

Moreover, rapid feedback on documentation is becoming crucial for preserving project pace as agile and DevOps approaches gain popularity. As teams move through development cycles, an automated checking and grading tool can be effortlessly incorporated into continuous integration/continuous delivery (CI/CD) pipelines to enable real-time validation of documentation quality. Other academic fields have looked into automating the grading process, especially for text assessments and essay grading. Although there are many technologies available to automate these procedures, there is still a lack of academic study on documentation for technical evaluation. Current approaches, such as peer evaluation platforms and automatic grading based on rubrics, fall short of addressing the particular difficulties presented by technical documents, which frequently contain both textual and diagrammatic components.

Furthermore, as organizations seek to scale their operations, automating documentation reviews allows businesses to maintain consistent quality across a growing number of projects. By leveraging machine learning algorithms trained on historical grading data, this project aims to automate and standardize the evaluation of SRS and SDD documents, providing consistent feedback while also reducing the workload for educators. By continuously improving the tool's accuracy, companies can ensure that their documentation standards evolve alongside the complexities of their software projects, ultimately leading to more robust, reliable systems.

1.2.2 Business

The proposed Automated Grading Tool for SRS and SDD documents fulfills a significant need by streamlining the evaluation process for technical documentation. In both academic and corporate settings, manually reviewing these documents can be inefficient, subjective, and error-prone. This tool automates the grading process, offering objective, detailed feedback based on established criteria such as clarity and adherence to requirements. It boosts productivity by reducing the time and effort required for evaluations, ensures consistency and fairness in grading, and can scale to accommodate large volumes of documents. Additionally, the tool seamlessly integrates with existing workflows, adapts to industry standards, and improves over time through the continuous learning capabilities of Large Language Models (LLMs) making it a practical and efficient solution for enhancing document review quality and speed.

1.3 Problem Statement

This project aims to address two primary challenges in the evaluation of technical documentation in software engineering education:

1. **Time-Intensive Review Process:** The manual grading of Software Requirements Specifications (SRS) and Software Design Descriptions (SDD) is labor-intensive, requiring significant time and effort from educators, especially in larger classes. This lengthy review process delays feedback, hindering students' ability to improve their documentation skills in real-time.

2. **Inconsistent Grading Practices:**The current manual evaluation of Software Requirements Specifications (SRS) and Software Design Descriptions (SDD) often leads to variability in grading due to differing standards and interpretations among instructors. This inconsistency can negatively impact the quality of feedback received by students.

By developing an Automated Checking and Grading Tool, this project seeks to significantly reduce manual labor and streamline the grading process, ultimately enhancing the educational experience for both students and instructors.

2 Project Description

2.1 Objectives

1. Automate Grading: Develop and implement an automated grading system for Software Requirements Specifications (SRS) and Software Design Descriptions (SDD) that can evaluate submissions based on predefined criteria, achieving at least 90% accuracy in grading within six months of deployment.

2. Provide Feedback: Create a feedback mechanism within the system that generates detailed, actionable feedback for each submission, allowing students to access their grades and feedback within 24 hours of submission, with professor approval.

3. Self-Learning Model: Integrate a self-learning algorithm that updates its grading criteria based on historical grading data and user feedback, resulting in a 20% improvement in grading consistency and accuracy over the first year of operation.

2.2 Scope

1. **Development of an Automated Grading Tool**: Create a software application that automates the grading of Software Requirements Specifications (SRS) and Software Design Descriptions (SDD) using Large Language Models (LLMs) to evaluate documents based on predefined criteria.

2. **Focus on Technical Documentation**: Specifically target the grading of SRS and SDD documents, addressing the unique challenges posed by technical content, including both textual and diagrammatic elements.

3. **Continuous Learning and Improvement**: Implement a self-learning mechanism that enables the grading model to evolve over time by learning from new data and feedback, enhancing its accuracy and effectiveness.

4. **User Training and Support**: Provide training and support for faculty members to effectively utilize the tool, ensuring they can interpret the results and feedback generated by the system.

5. **Feedback Mechanism**: Develop a robust feedback system that not only grades the documents but also provides detailed insights into areas of strength and opportunities for improvement.

6. **Evaluation Criteria**: Define clear evaluation criteria for the grading process, focusing on aspects such as completeness, clarity, adherence to business needs, and overall structure of the documentation.

7. **Future Scalability**: Design the system with scalability in mind, allowing it to be adapted for other types of academic documentation in the future.

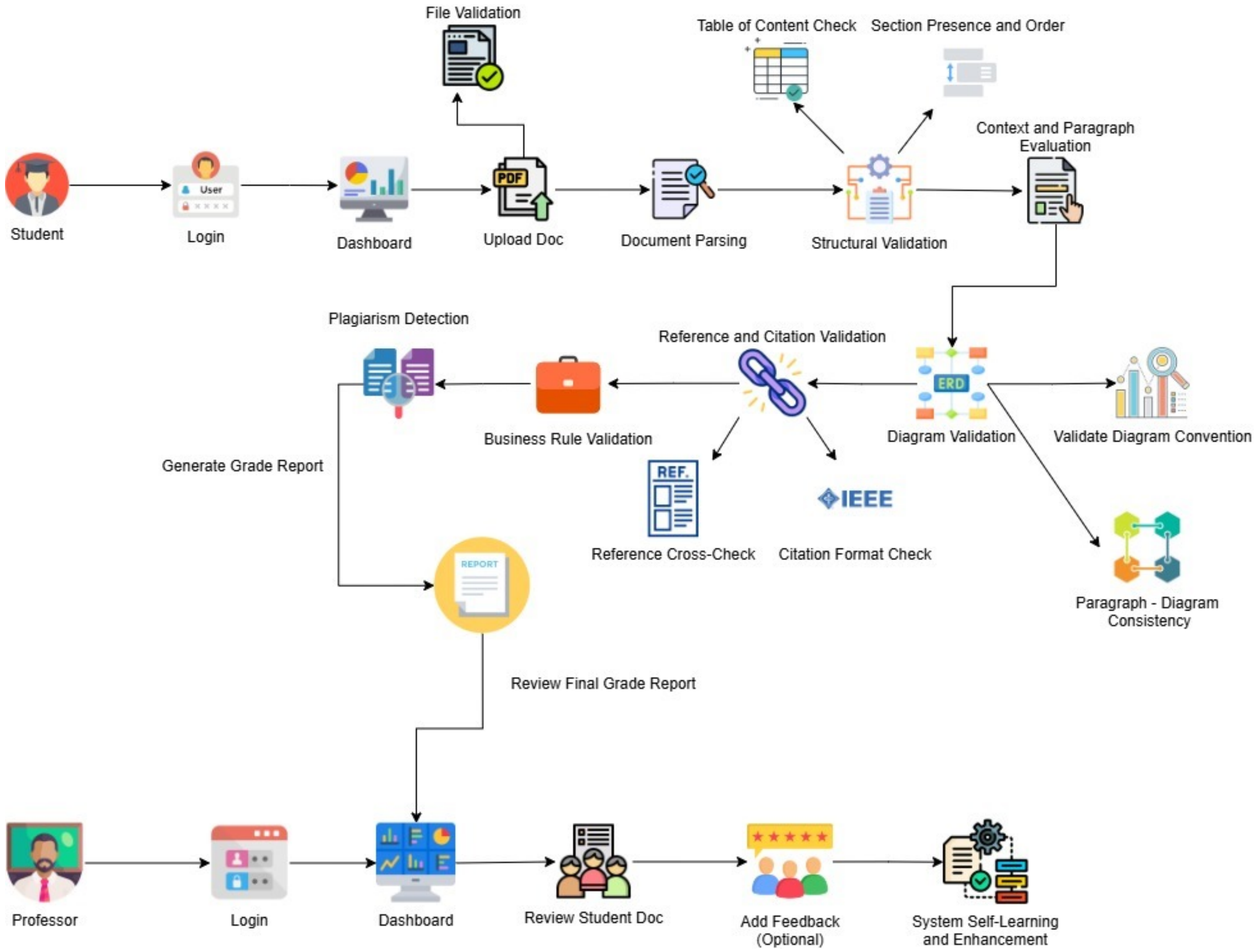


Figure 1: Project Overview

2.3 Project Overview

As shown in Figure 1, the student logs in to the system and submits their Software Requirements Specification (SRS) or Software Design Description (SDD). Once submitted, the system analyzes the document using a Large Language Model (LLM) trained to evaluate it against criteria, such as clarity, completeness, and adherence to business needs. The LLM then generates an initial grade and detailed feedback report, highlighting strengths and areas for improvement.

Next, the grade and feedback are held for approval by the professor. The student can only view their results after receiving approval, ensuring that the feedback is discussed and understood in the academic context.

Meanwhile, the professor logs into the system to access the submitted documents. They can view all SRS and SDD submissions along with their respective grades and feedback. Professors have the option to review and modify the system-generated feedback before approving it for student access. Finally, the professor can download comprehensive reports summarizing grades and feedback for administrative purposes.

2.4 Stakeholder

2.4.1 Internal

- Zeina Hesham: (Team Leader)
- George Ayman: (Team member)
- Nouran Mohamed: (Team member)
- Nader Amir: (Team member)

2.4.2 External

- **Students**: The primary end users who submit their Software Requirements Specifications (SRS) and Software Design Descriptions (SDD) to the system for grading. They receive feedback and grades, which are accessible only with the professors' approval.
- **Professors**: While they are also end users, professors can be considered clients in the context of this project because they require the system to facilitate their grading process and provide insights into student submissions.

3 Similar System

3.1 Academic

Chu et al.[1] tackle the pressing issue of grading open-ended short-answer questions (SAQs) in educational contexts, a process that is often labor-intensive and susceptible to inconsistencies due to subjective interpretations by instructors. The main problem statement of their work highlights the need for a more efficient and reliable grading system that can provide timely feedback while minimizing biases. To address this challenge, the researchers introduced GradeOpt, an innovative multi-agent framework that leverages large language models (LLMs) to enhance the automatic grading process. This framework not only incorporates LLMs for grading but also integrates additional agents that facilitate reflection and refinement of the grading criteria, thereby aligning the evaluations more closely with human assessors. The dataset used in their research consisted of responses from middle school mathematics teachers, specifically designed to evaluate their content knowledge (CK) and pedagogical content knowledge (PCK). The results of their experiments revealed that GradeOpt significantly improved grading accuracy and consistency compared to traditional methods, demonstrating its effectiveness in providing timely and reliable evaluations in educational assessments. Overall, their work represents a substantial contribution to the field of automatic grading, offering a promising solution to enhance the quality of feedback in learning analytics.

Vijaya Shetty Sadanand[2] and her team developed an automated essay evaluation system that addresses the challenges of manual grading, which is often time-consuming and inconsistent. The main problem statement of their work was to create a reliable and efficient grading system that could provide timely feedback to students while ensuring accuracy. To tackle this issue, the researchers proposed a long short-term memory (LSTM) model trained on a dataset of 12,000 manually graded essays, which allowed the system to learn and generate grading parameters effectively. Additionally, they employed sentiment analysis using a Naïve Bayes classifier trained on a Twitter dataset of 10,000 tweets to assess the sentiment of the essays. The main results of their research demonstrated the system's effectiveness, achieving an average quadratic weighted kappa (QWK) score of 0.911 for the grading model and a remarkable 99.4% accuracy for the sentiment analysis classifier. As illustrated in Figure of their paper, the architecture of the automated essay grading system encompasses various components, including feature extraction, sentiment analysis, and plagiarism detection, which collectively contribute to the final grading process. This comprehensive approach not only automated the grading process but also provided students with detailed feedback, including corrections for syntactic errors and plagiarism checks, thereby enhancing the overall educational experience.

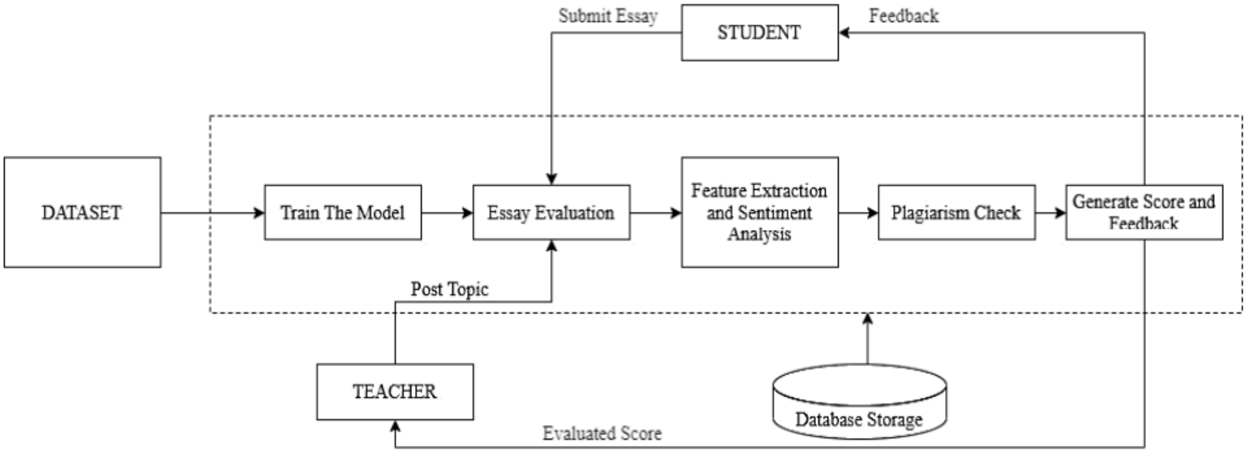


Figure 2: *Automated scoring and feedback systems*[2]

Julio C. Caiza and Jose M. Del Alamo [3] provide a comprehensive review of the state of automatic grading tools for programming assignments, identifying the main problem statement as the lack of a common grading model across various tools, which leads to inconsistencies in assessment. They contribute to solving this problem by analyzing existing tools and proposing a grading metrics characterization as a foundational step towards establishing a standardized grading model. The researchers utilized a dataset comprising various mature and recent tools, examining their features and functionalities to draw comparisons. The main results indicate significant advancements in areas such as security, integration with Learning Management Systems (LMS), and plagiarism detection, while still highlighting the ongoing challenge of achieving a unified grading approach. However, the paper could be criticized for not providing enough empirical data to support the claims made about the effectiveness of the tools reviewed, as well as for lacking a detailed exploration of user experiences and feedback from educators who implement these tools in real-world settings.

Amira A. Alshazly[4]addresses the main problem of inadequate defect detection techniques in software requirements specifications (SRS), highlighting the limitations of existing taxonomies that fail to correlate defects with their causes. To tackle this issue, the researchers proposed a comprehensive taxonomy of requirements defects and introduced a combined-reading technique to enhance defect detection efficiency. They conducted experiments using four SRS case studies to evaluate the effectiveness of various reading techniques, including Checklist-Based Reading (CBR), Scenario-Based Reading (PBR), and Defect-Based Reading (DBR). The dataset comprised four distinct SRS documents, which were analyzed to identify and classify defects. The main results revealed a total of 417 distinct defects across six types, confirming the reliability of the proposed taxonomy and demonstrating the effectiveness of the combined reading technique in improving the quality of software requirements.

Mahana, Jons et al. [5] addressed the challenge of automating the grading process for theoretical exam answers in technical education, a field traditionally reliant on human evaluators due to the complexity and variability of student responses. Their work focuses on the inefficiencies and biases of traditional grading methods, which often result in inconsistencies and a heavy workload for educators. To tackle this, the researchers developed a machine learning framework that utilizes natural language processing (NLP) techniques, including the bag of words model, tf-idf patterns, and semantic analysis, to analyze and grade student responses. They constructed a dataset from minor exam papers of B.Tech. courses, which, although limited in scope, provided a basis for their experiments and enabled them to evaluate the effectiveness of their grading algorithms. The results showed that the model performed best when applied to descriptive theoretical questions, particularly those requiring detailed explanations and definitions. This highlights the importance of content specificity in technical domains, suggesting that essays with a minimum word count and comprehensive responses yield more accurate grading outcomes. Overall, their research demonstrates the feasibility of automating grading in technical education and underscores the potential of machine learning to improve grading efficiency, objectivity, and scalability.

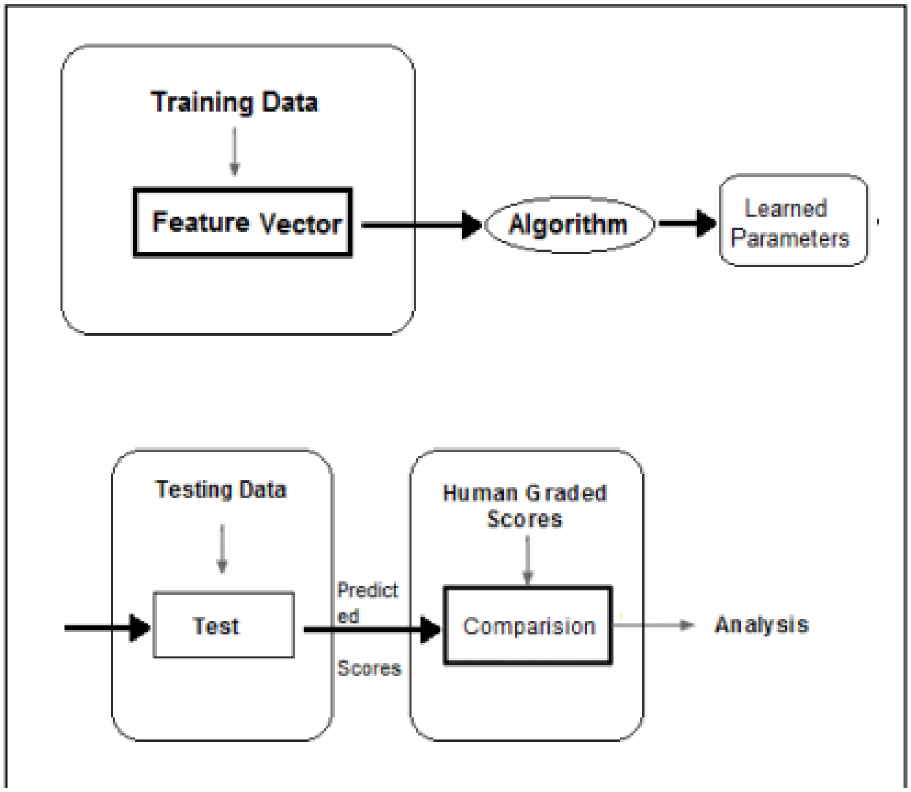


Figure 3: *Automated content grading using machine learning*[5]

Syed Waqas and Imran Shafi [6] addresses the critical issue of inadequacies in Software Requirement Specification (SRS) documents in their work, "Process to Enhance the Quality of Software Requirement Specification Document." The main problem statement highlights how these inadequacies, characterized by ambiguity, incompleteness, and inconsistency, significantly contribute to the failure of software projects. To solve this pressing problem, the researchers proposed a comprehensive methodology that includes four key processes: Parsing Requirement, Requirement Mapping using Matrix, Addition of Requirements in SRS template, and Third Party Inspection. This structured approach aims to enhance the quality of SRS documents by ensuring completeness and correctness from multiple stakeholder perspectives. While the paper does not explicitly detail the dataset used for their analysis, it indicates that the methodology was implemented in a practical setting within a software house, allowing for real-world application and evaluation. The main results of their research demonstrate significant improvements in the quality of SRS documents, particularly in terms of requirement completeness and correctness, suggesting that their proposed approach effectively mitigates the quality issues that have historically plagued SRS documentation and contributes positively to the success of software projects.

Maninder Singh[7] addresses the critical problem of accurately identifying true positives and false positives in software requirement reviews, which is essential for improving the quality of software development. To tackle this issue, the researcher proposed a machine learning framework that utilizes natural language processing (NLP) techniques to analyze inspection reviews and classify them effectively. The dataset used in this research consisted of natural language inspection reviews collected from multiple software requirements specifications (SRS) documents, which served as both training and testing sets for the classification models. The main results of the study demonstrated that the implemented classification methods significantly improved the accuracy of identifying fault-prone requirements, thereby providing valuable insights for requirement authors to enhance the quality of their documentation. Overall, Singh's work contributes to the field by offering a systematic approach to automating the validation of requirement reviews, ultimately leading to more reliable software development processes.

Oluwole Fagbohun, along with Nwaamaka Pearl Iduwe and Mustapha Abdullahi [8]identified the primary issue in their research as the limitations of traditional grading systems, which often suffer from scalability issues, inconsistency in evaluation, and a lack of personalized feedback for students. These systems, reliant on manual grading, are time-consuming and prone to subjective bias, especially when applied on a large scale. To address this problem, the researchers explored the integration of large language models (LLMs) into the grading process. By leveraging the advanced natural language processing (NLP) capabilities of LLMs, they developed an automated grading system that could assess complex student responses, such as short answers and essays, while providing more detailed and consistent feedback. The dataset utilized in their study consisted of a range of student responses across different formats, allowing the LLMs to be tested on various types of academic work. The main results demonstrated that LLMs significantly improved the accuracy, fairness, and efficiency of the grading process. However, the researchers also noted the importance of addressing challenges such as potential biases in AI models and emphasized the need for ongoing human oversight to ensure fairness and reliability in the automated grading system.

Marcus Messer, Neil C. C. Brown, Michael Kölling, and Miaojing Shi[9] conducted a systematic literature review to analyze and evaluate automated grading and feedback tools for programming education, reviewing 121 relevant papers from an initial dataset of 1,490 sourced from IEEExplore, ACM Digital Library, and Scopus, and categorizing the tools based on the skills assessed, evaluation techniques employed, and supported programming languages. While many tools were effective in assessing code correctness, the study highlighted significant gaps in feedback related to code maintainability and readability, underscoring the need for improved evaluation methods to help educators select suitable tools, ultimately enhancing the learning experience for students.

Gloria Ashiya Katuka[10] and her colleagues conducted a comprehensive investigation into the application of Large Language Models (LLMs) for automatic scoring and feedback generation, focusing on the critical challenge of improving the accuracy and efficiency of grading student responses in educational settings. The main problem statement of their work centered on the limitations of traditional grading methods, which often struggle to provide timely and personalized feedback, particularly in online learning environments where instructor resources are constrained. To address this issue, the researchers contributed by fine-tuning various pretrained transformer models, including RoBERTa, GPT-2, and LLaMA-2, employing a proprietary dataset that consisted of assessment questions, student answers, and corresponding graded scores and feedback across multiple subjects. Their rigorous experiments demonstrated that the LLaMA-2 model, especially when enhanced with grade scores as additional input, achieved the highest performance in terms of similarity to expert feedback. The results indicated significant improvements in grading accuracy, with the models achieving less than 3% error on average, thereby showcasing the potential of LLMs to augment the grading process and provide valuable insights for both educators and students in online learning environments. This work not only highlights the effectiveness of LLMs in educational technology but also paves the way for future advancements in automated assessment systems.

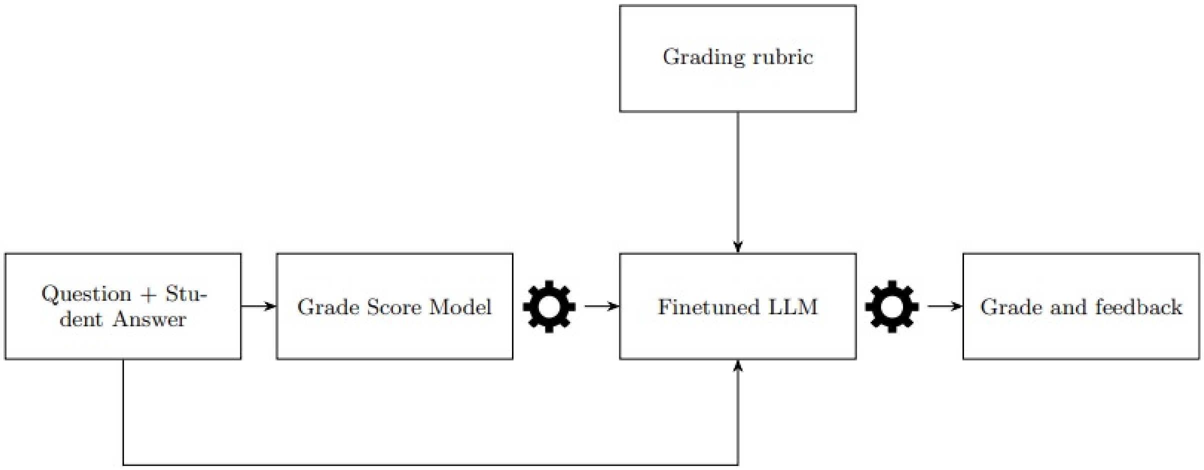


Figure 4: *Investigating Automatic Scoring and Feedback using Large Language Models*[10]

Humasak Simanjuntak's [11]research focuses on developing an Automatic Grading System (AGS) for Entity Relationship (ER) Diagrams, addressing the significant problem of inconsistent and inefficient grading practices in educational settings, particularly due to the high volume of submissions and the common occurrence of similar errors among students. To contribute to solving this issue, the researchers proposed a comprehensive framework that employs two distinct approaches: the first utilizes the Tree Edit Distance algorithm to assess the similarity of ER Diagrams represented in XML Metadata Interchange (XMI) format, generating similarity scores and feedback for students; the second approach leverages machine learning algorithms to create a classifier that can automatically grade ER Diagrams based on features extracted from the XMI files. The dataset used in this study consists of XMI files from student submissions and expert-graded solutions, allowing for a robust evaluation of the grading system. The main results indicate that the proposed framework can deliver accurate and consistent assessments, providing valuable feedback to students and enhancing the grading process. This research highlights the potential of integrating technology into educational assessment, paving the way for more efficient and standardized grading systems in database design education.

3.2 Business Applications

Turnitin[12] is a widely used plagiarism detection tool designed to ensure the originality of academic and professional content. It is commonly used in academic institutions, businesses, and corporate settings to validate written work, including essays, reports, and technical documents. Turnitin helps detect instances of plagiarism by comparing submitted content against a vast database of academic papers, articles, websites, and other documents

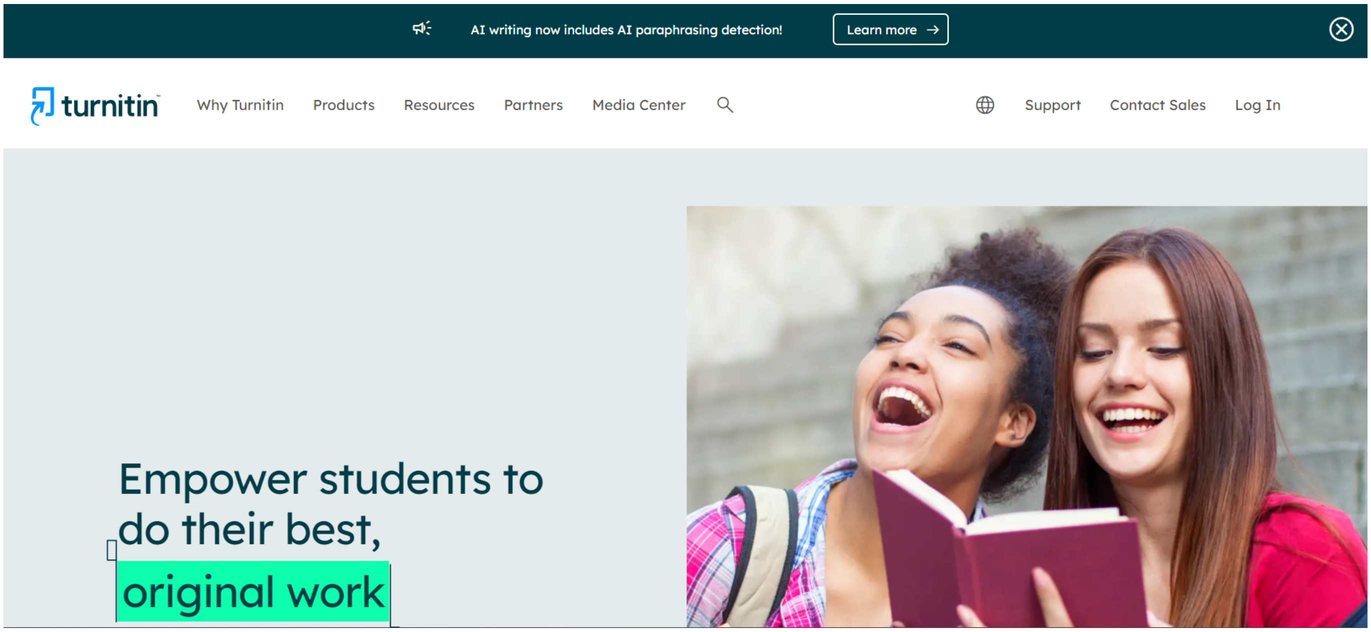

AI writing now includes AI paraphrasing detection!

Figure 5: Turnitin

Universities are a key example of how automated grading tools can be applied effectively in a business context. In academic environments, especially within technical fields like software engineering, students are required to produce complex documentation such as Software Requirements Specifications (SRS) and Software Design Descriptions (SDD). Manually grading these documents is both time-intensive and subject to variability. An automated grading tool can streamline this process, ensuring faster, more consistent evaluations, and providing detailed feedback. Beyond academia, this technology can be applied in various industries that rely on documentation, such as business organizations, consulting firms, and government agencies, where consistent and accurate document evaluation is critical. By adopting automated grading tools, these organizations can improve productivity, reduce human error, and ensure compliance.

4 What is new in the Proposed Project?

Feature	Manual Grading	Proposed Automated System
Time and Effort	Time-consuming, especially for large classes. Manual review of each document takes significant effort.	Reduces grading time significantly by automating document evaluation, allowing for quicker feedback.
Consistency	Highly subjective, grading can vary between instructors and even within the same instructor over time.	Provides consistent, objective results based on predefined Large Language Models (LLMs) and evaluation criteria.
Evaluation and Precision	Prone to human error and inconsistencies in applying grading rubrics.	Improved accuracy through predefined evaluation criteria, Large Language Models (LLMs), and their continuous learning capabilities.
Scalability	Limited scalability. Grading becomes increasingly difficult with more students and documents.	Scales easily to accommodate larger volumes of documents and can be adapted to other types of technical documentation.
Adaptability to Feedback	Instructors can adjust feedback manually but may not consistently improve grading practices.	Includes a self-learning model that incorporates professor feedback, improving grading accuracy over time.
Diagram Evaluation	Requires manual checking for the correctness and relevance of diagrams, which is time-consuming.	Automates the evaluation of diagrams based on predefined criteria, improving both speed and accuracy.

Table 2: Comparison between Manual Grading and Proposed Automated System

5 Proof of concept

The image below highlights the key functionalities we have implemented till now in the project. Below are screenshots of the code that demonstrate the proof of concept for the extraction and structuring process.

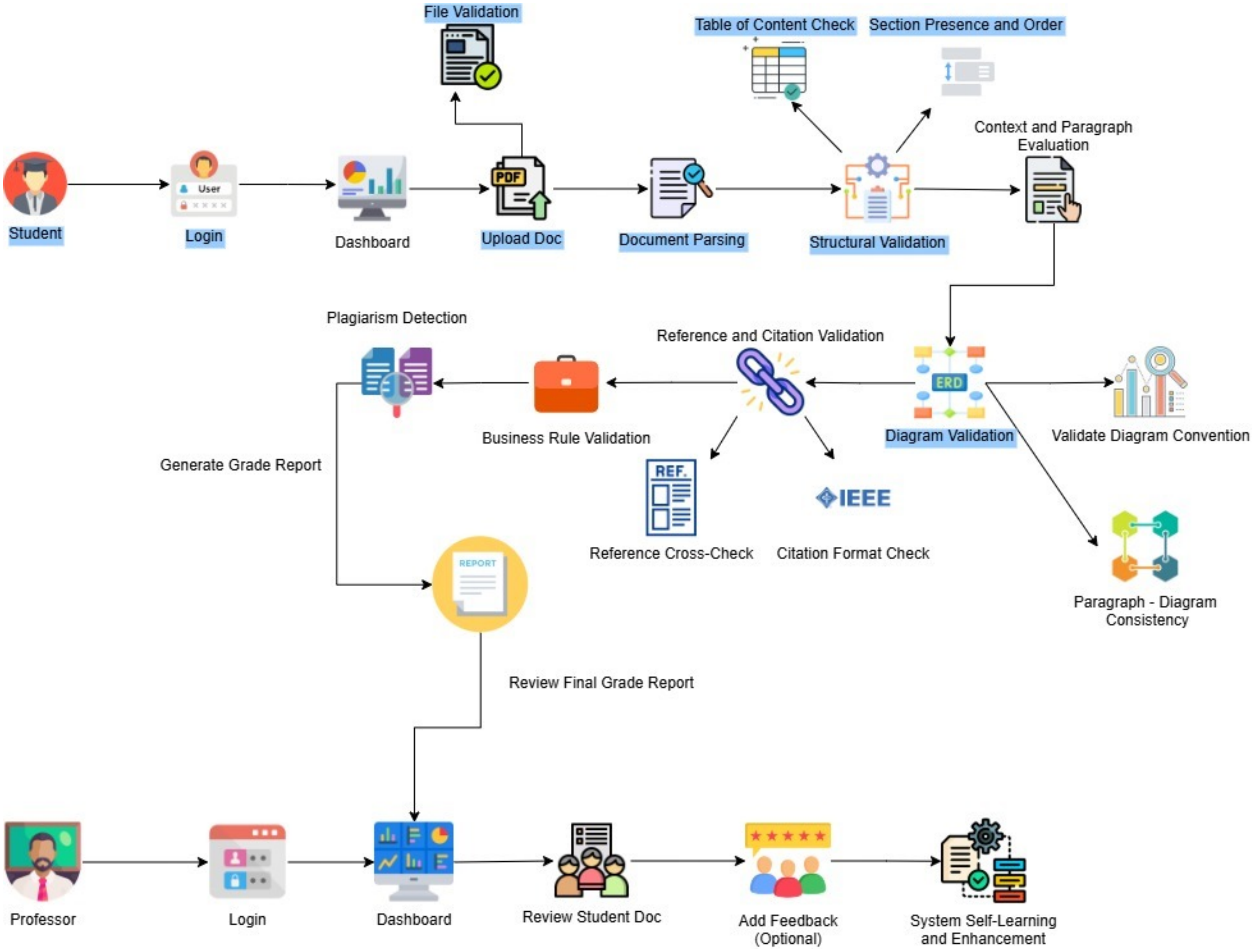
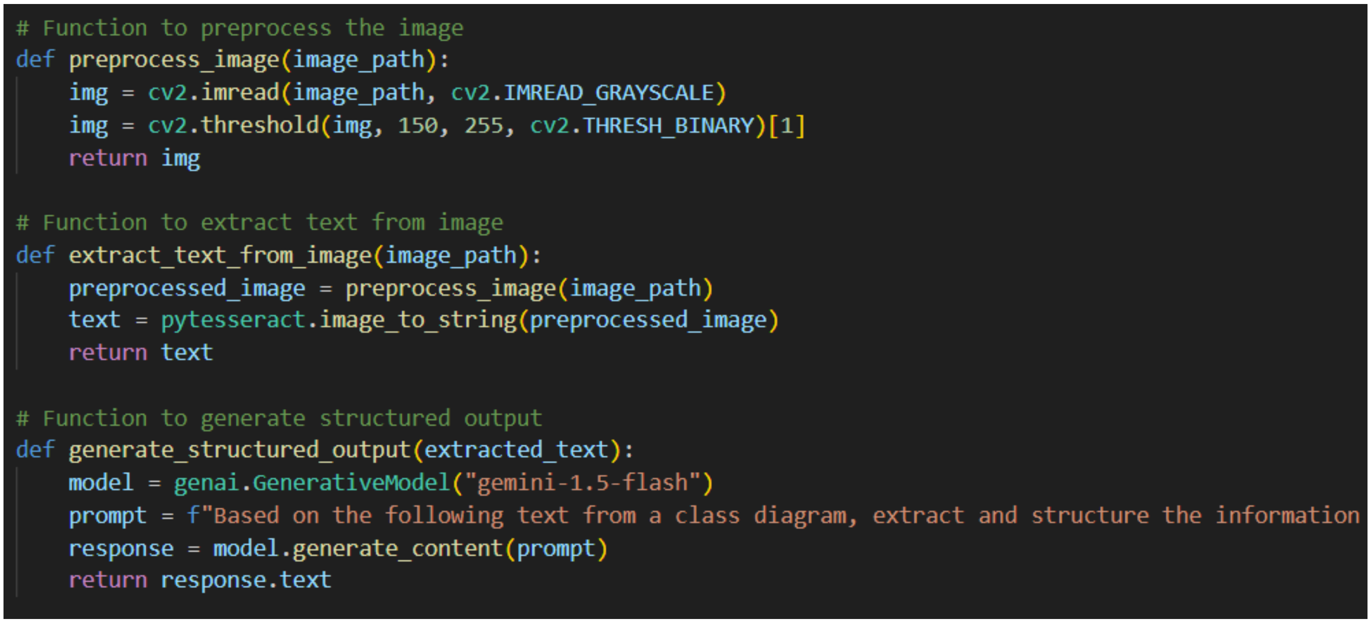


Figure 6: Project Overview Implemented

The code below shows how we extract text from an image using OCR in Python. It involves three main functions:

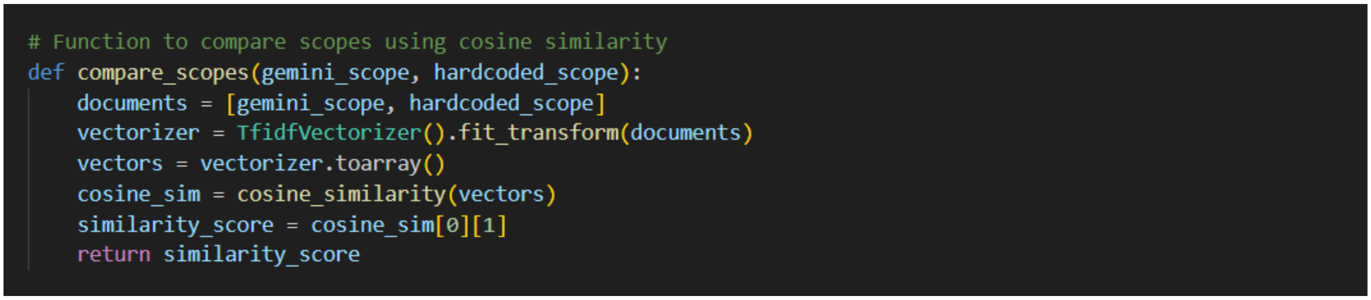
- `preprocess_image(image_path)`: Converts the image to grayscale and applies binary thresholding.
- `extract_text_from_image(image_path)`: Uses `pytesseract` to extract text after preprocessing.
- `generate_structured_output(extracted_text)`: Structures the extracted text using a Generative AI model (gemini-1.5-flash) for diagrams.



```
# Function to preprocess the image
def preprocess_image(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    img = cv2.threshold(img, 150, 255, cv2.THRESH_BINARY)[1]
    return img

# Function to extract text from image
def extract_text_from_image(image_path):
    preprocessed_image = preprocess_image(image_path)
    text = pytesseract.image_to_string(preprocessed_image)
    return text

# Function to generate structured output
def generate_structured_output(extracted_text):
    model = genai.GenerativeModel("gemini-1.5-flash")
    prompt = f"Based on the following text from a class diagram, extract and structure the information
    response = model.generate_content(prompt)
    return response.text
```

Figure 7: Extract Text From Image

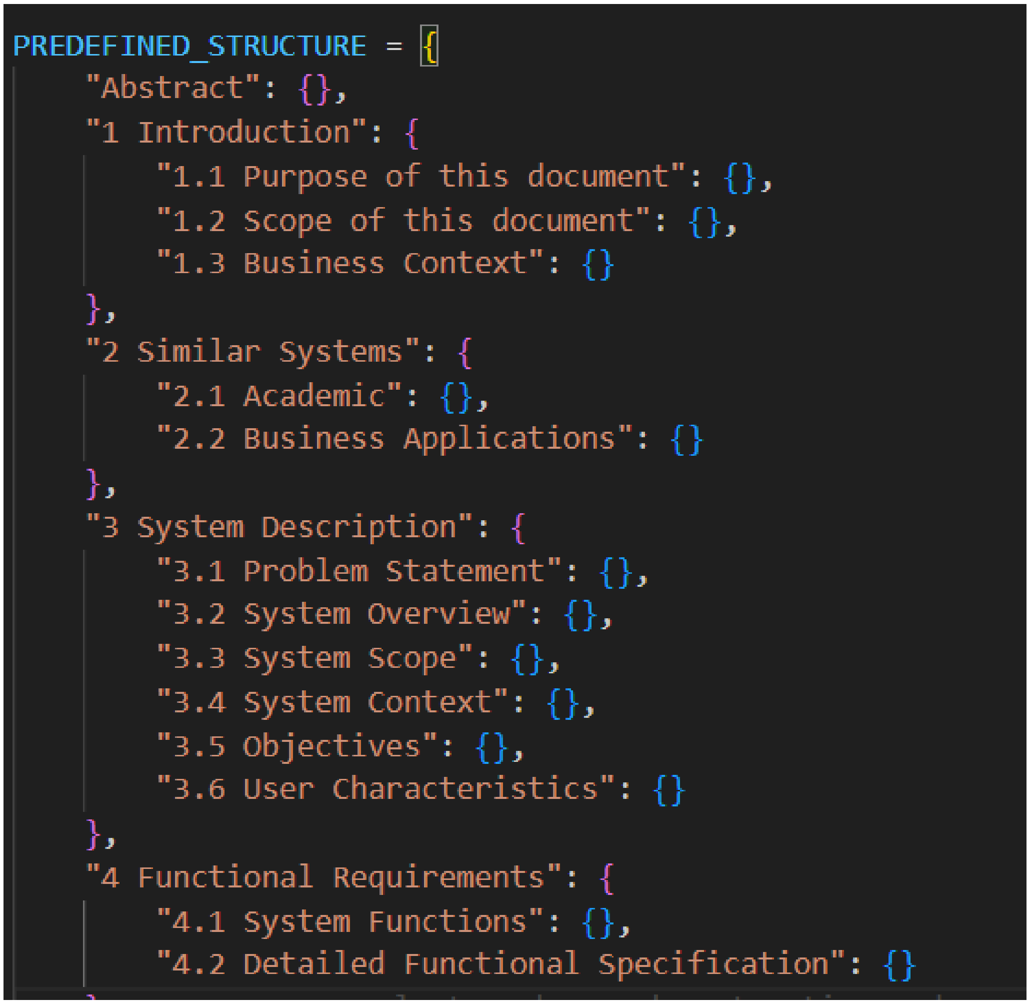
The code compares two text-based scopes using cosine similarity. It first converts the scopes into numerical vectors with `TfidfVectorizer` from `sklearn`, then computes the cosine similarity between the vectors using `cosine_similarity`. The resulting similarity score, ranging from 0 to 1, is extracted from the similarity matrix, where 1 indicates identical scopes and 0 indicates no similarity.



```
# Function to compare scopes using cosine similarity
def compare_scopes(gemini_scope, hardcoded_scope):
    documents = [gemini_scope, hardcoded_scope]
    vectorizer = TfidfVectorizer().fit_transform(documents)
    vectors = vectorizer.toarray()
    cosine_sim = cosine_similarity(vectors)
    similarity_score = cosine_sim[0][1]
    return similarity_score
```

Figure 8: Cosine Similarity

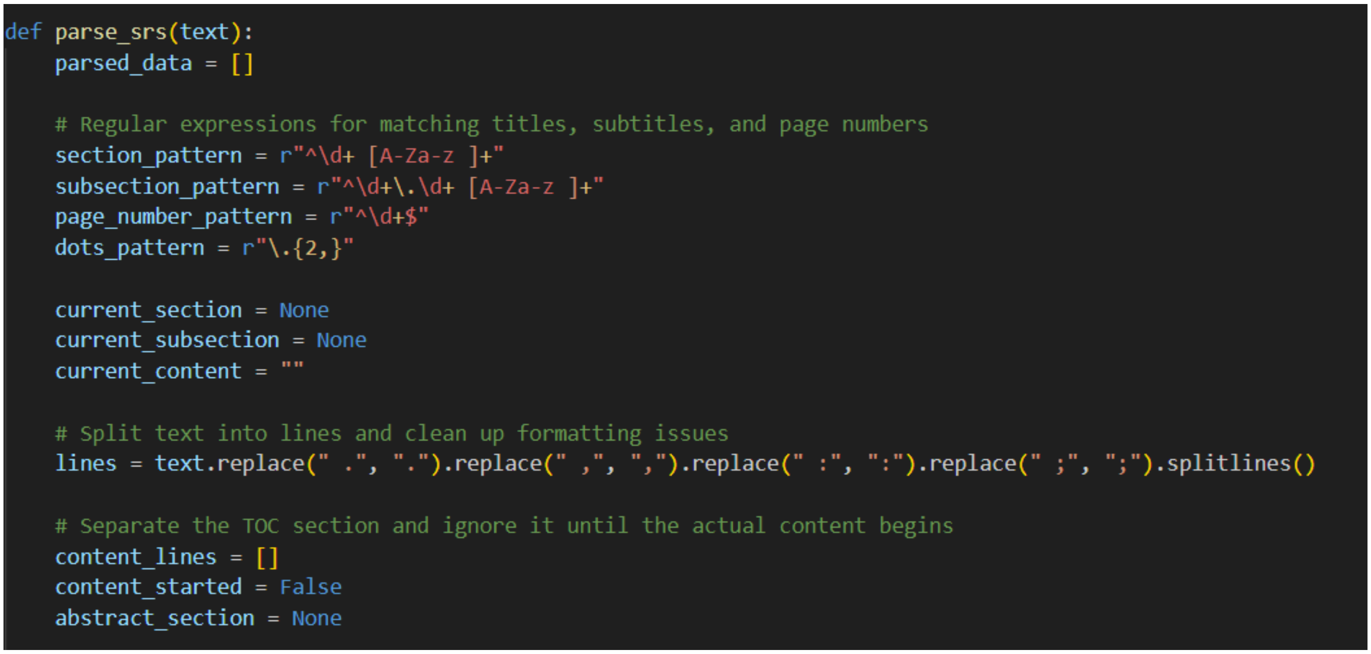
The following is the predefined structure used in our Software Requirements Specification (SRS) document. This structure serves as a guideline to ensure that any SRS input follows the required format and contains the necessary sections and subsections. By adhering to this structure, we can maintain consistency and completeness across all SRS documents.



```
PREDEFINED_STRUCTURE = {
    "Abstract": {},
    "1 Introduction": {
        "1.1 Purpose of this document": {},
        "1.2 Scope of this document": {},
        "1.3 Business Context": {}
    },
    "2 Similar Systems": {
        "2.1 Academic": {},
        "2.2 Business Applications": {}
    },
    "3 System Description": {
        "3.1 Problem Statement": {},
        "3.2 System Overview": {},
        "3.3 System Scope": {},
        "3.4 System Context": {},
        "3.5 Objectives": {},
        "3.6 User Characteristics": {}
    },
    "4 Functional Requirements": {
        "4.1 System Functions": {},
        "4.2 Detailed Functional Specification": {}
```

Figure 9: Predefined Structure of SRS Document

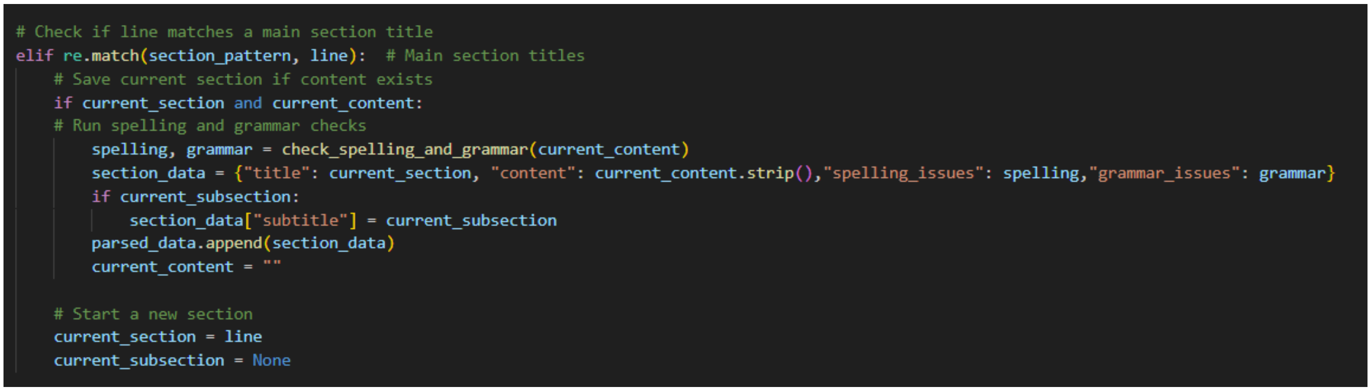
The following Python code defines a function `parse_srs` to extract and organize sections and subsections from a Software Requirements Specification (SRS) document. The function aims to parse the text of an SRS document and structure it in a way that can be further processed or analyzed.



```python
def parse_srs(text):
    parsed_data = []

    # Regular expressions for matching titles, subtitles, and page numbers
    section_pattern = r"^\d+ [A-Za-z ]+"
    subsection_pattern = r"^\d+\.\d+ [A-Za-z ]+"
    page_number_pattern = r"^\d+$"
    dots_pattern = r"\.{2,}"

    current_section = None
    current_subsection = None
    current_content = ""

    # Split text into lines and clean up formatting issues
    lines = text.replace(" .", ".").replace(" ,", ",").replace(" :", ":").replace(" ;", ";").splitlines()

    # Separate the TOC section and ignore it until the actual content begins
    content_lines = []
    content_started = False
    abstract_section = None
```

Figure 10: Spliting Document into Sections

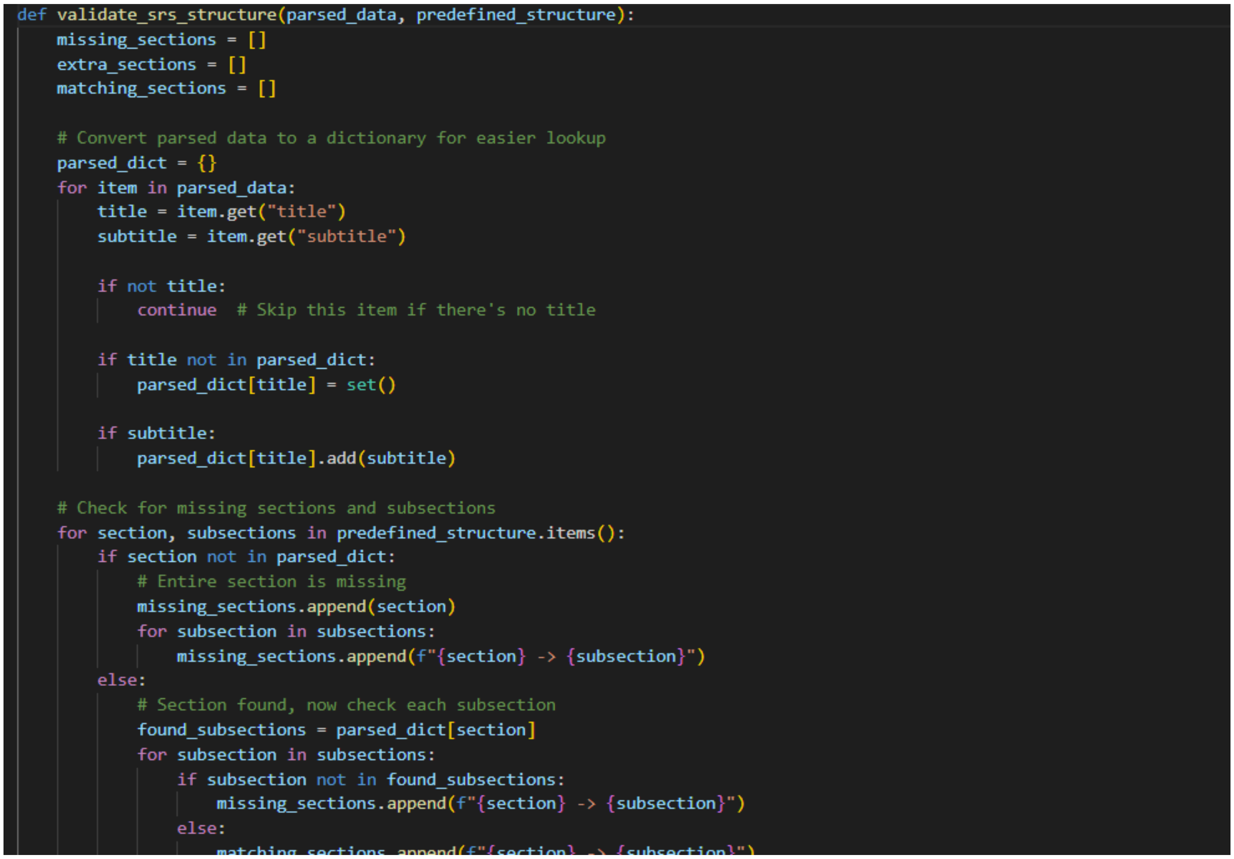
The below Python code adds a layer of functionality to perform spelling and grammar checks and structures the section data for further analysis.



```python
# Check if line matches a main section title
elif re.match(section_pattern, line):  # Main section titles
    # Save current section if content exists
    if current_section and current_content:
    # Run spelling and grammar checks
        spelling, grammar = check_spelling_and_grammar(current_content)
        section_data = {"title": current_section, "content": current_content.strip(),"spelling_issues": spelling,"grammar_issues": grammar}
        if current_subsection:
            section_data["subtitle"] = current_subsection
        parsed_data.append(section_data)
        current_content = ""

    # Start a new section
    current_section = line
    current_subsection = None
```

Figure 11: Grammar and spelling check

The below function `validate_srs_structure` is used to validate whether the parsed SRS document adheres to a predefined structure. It identifies missing sections and subsections, as well as checks if extra sections have been added.



```python
def validate_srs_structure(parsed_data, predefined_structure):
    missing_sections = []
    extra_sections = []
    matching_sections = []

    # Convert parsed data to a dictionary for easier lookup
    parsed_dict = {}
    for item in parsed_data:
        title = item.get("title")
        subtitle = item.get("subtitle")

        if not title:
            continue  # Skip this item if there's no title

        if title not in parsed_dict:
            parsed_dict[title] = set()

        if subtitle:
            parsed_dict[title].add(subtitle)

    # Check for missing sections and subsections
    for section, subsections in predefined_structure.items():
        if section not in parsed_dict:
            # Entire section is missing
            missing_sections.append(section)
            for subsection in subsections:
                missing_sections.append(f"{section} -> {subsection}")
        else:
            # Section found, now check each subsection
            found_subsections = parsed_dict[section]
            for subsection in subsections:
                if subsection not in found_subsections:
                    missing_sections.append(f"{section} -> {subsection}")
                else:
                    matching_sections.append(f"{section} -> {subsection}")
```

Figure 12: Compare Parsed text with Predefined Structure

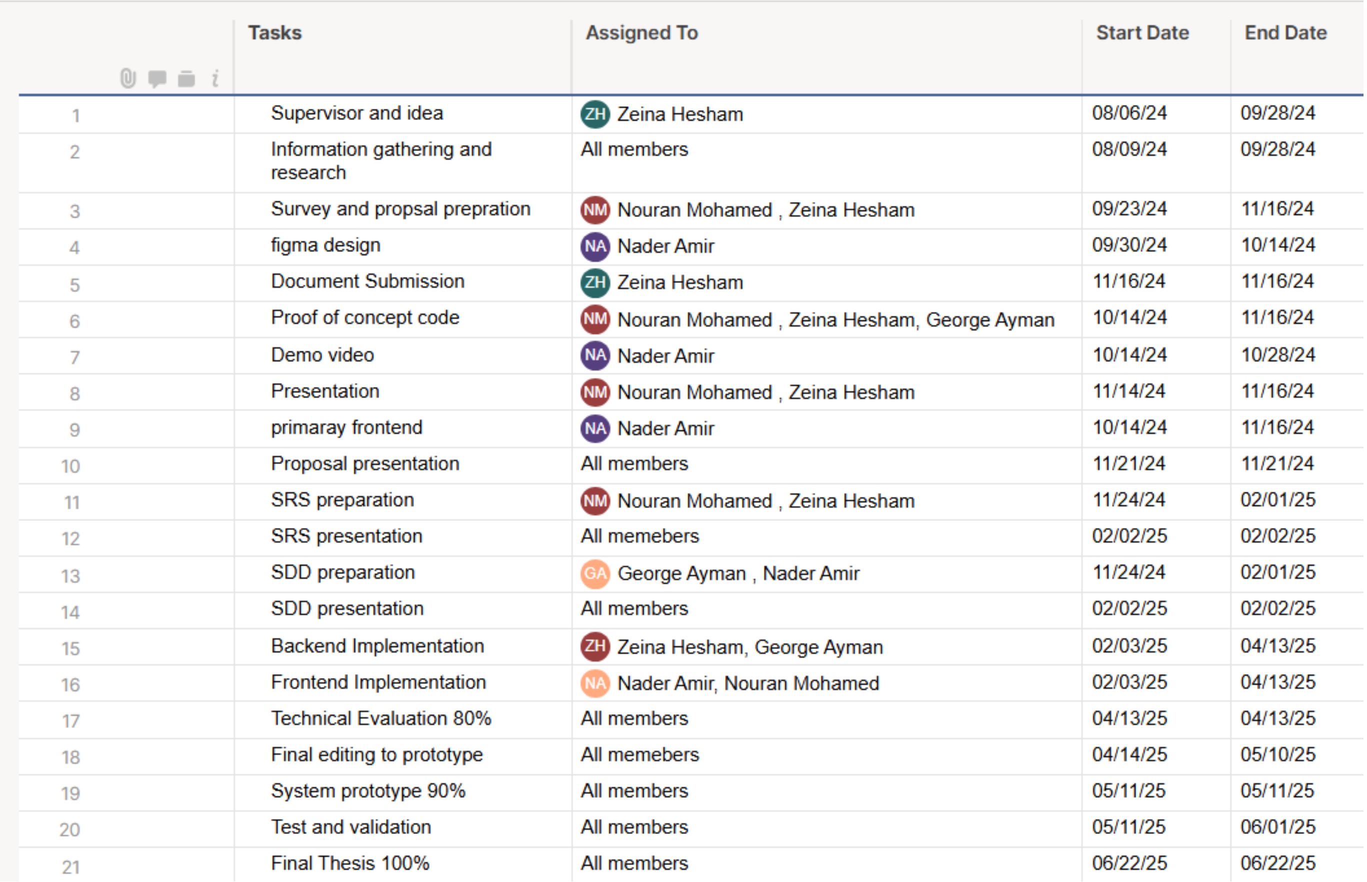
6 Project Management and Deliverables

6.1 Deliverables

This project's main objective is to automate and streamline the grading process for technical documentation, particularly Software Requirements Specifications (SRS) and Software Design Descriptions (SDD). The goal is to enhance grading efficiency, consistency, and objectivity by utilizing Large Language Models (LLMs) and natural language processing (NLP) technologies.

- Software proposal document.
- Software requirements specification.
- Software design description.
- Prototype Application
- Thesis document.
- Automated grading tool.
- Grading and feedback reports.

6.2 Tasks and Time Plan



	Tasks	Assigned To	Start Date	End Date
1	Supervisor and idea	ZH Zeina Hesham	08/06/24	09/28/24
2	Information gathering and research	All members	08/09/24	09/28/24
3	Survey and propsal prepration	NM Nouran Mohamed , Zeina Hesham	09/23/24	11/16/24
4	figma design	NA Nader Amir	09/30/24	10/14/24
5	Document Submission	ZH Zeina Hesham	11/16/24	11/16/24
6	Proof of concept code	NM Nouran Mohamed , Zeina Hesham, George Ayman	10/14/24	11/16/24
7	Demo video	NA Nader Amir	10/14/24	10/28/24
8	Presentation	NM Nouran Mohamed , Zeina Hesham	11/14/24	11/16/24
9	primaray frontend	NA Nader Amir	10/14/24	11/16/24
10	Proposal presentation	All members	11/21/24	11/21/24
11	SRS preparation	NM Nouran Mohamed , Zeina Hesham	11/24/24	02/01/25
12	SRS presentation	All memebers	02/02/25	02/02/25
13	SDD preparation	GA George Ayman , Nader Amir	11/24/24	02/01/25
14	SDD presentation	All members	02/02/25	02/02/25
15	Backend Implementation	ZH Zeina Hesham, George Ayman	02/03/25	04/13/25
16	Frontend Implementation	NA Nader Amir, Nouran Mohamed	02/03/25	04/13/25
17	Technical Evaluation 80%	All members	04/13/25	04/13/25
18	Final editing to prototype	All memebers	04/14/25	05/10/25
19	System prototype 90%	All members	05/11/25	05/11/25
20	Test and validation	All members	05/11/25	06/01/25
21	Final Thesis 100%	All members	06/22/25	06/22/25

Figure 13: Time Plan table

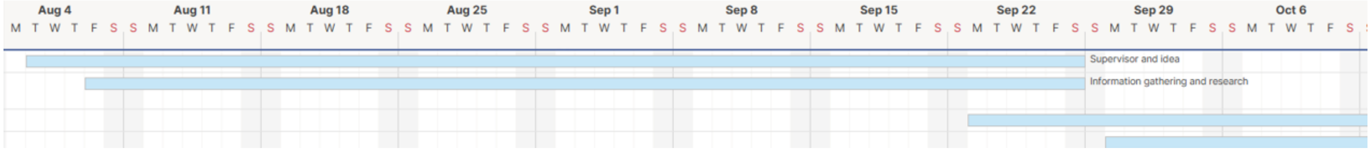


Figure 14: Time plan chart

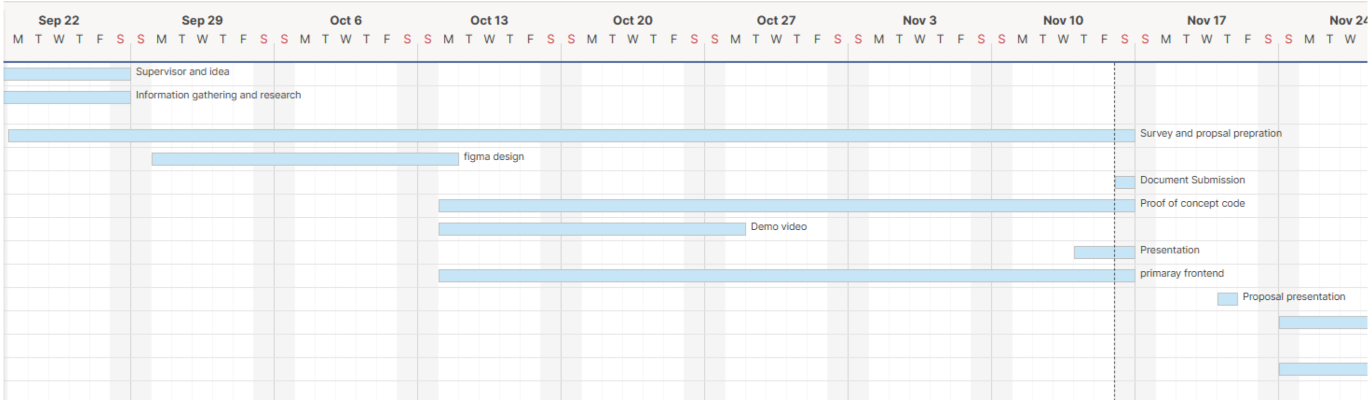


Figure 15: Time plan chart

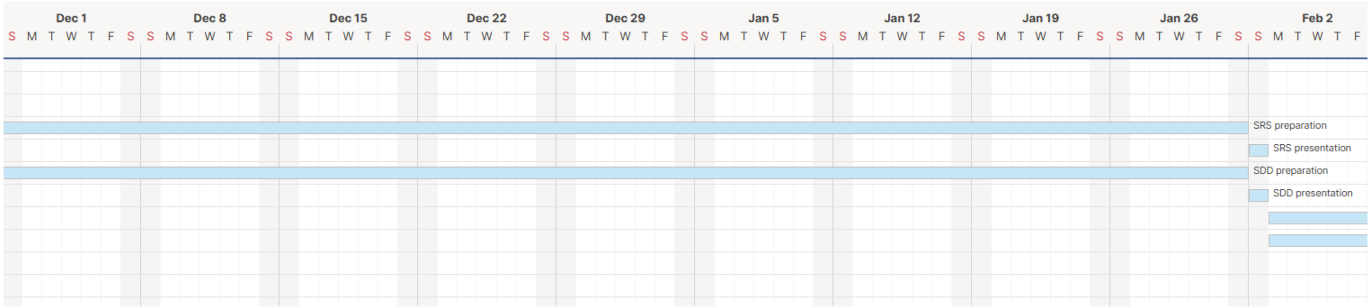


Figure 16: Time plan chart

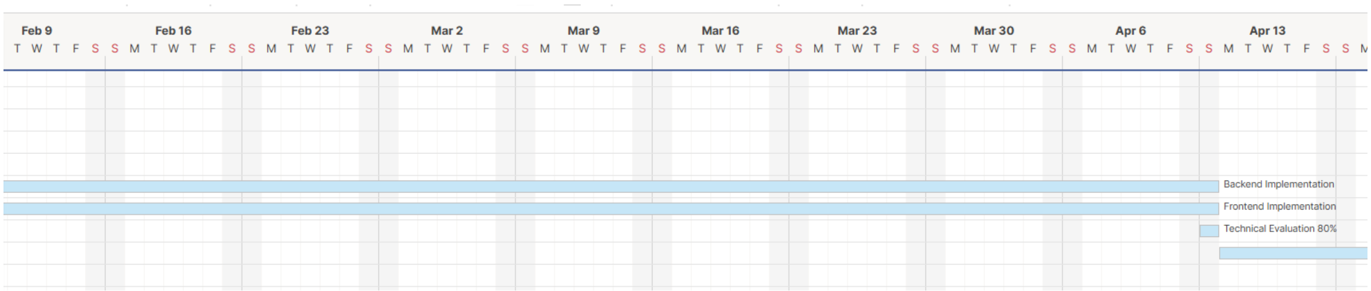


Figure 17: Time plan chart

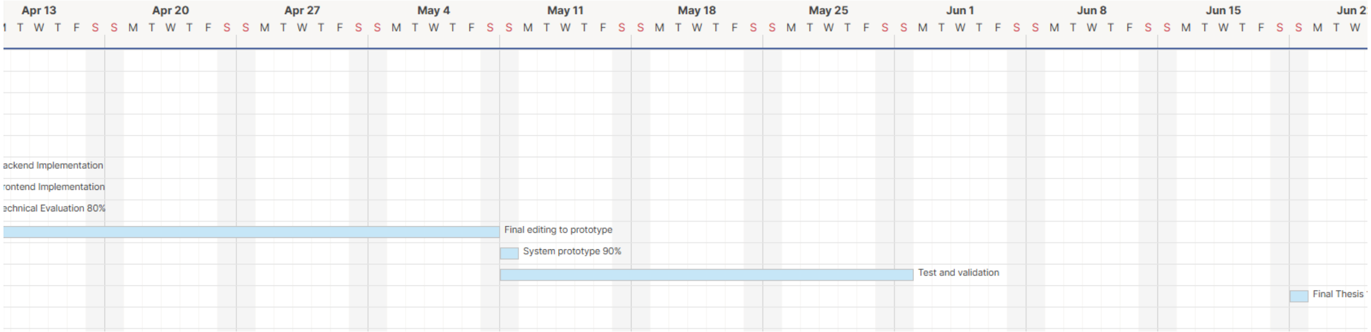


Figure 18: Time plan chart

6.3 Budget and Resource Costs

The budget for this project includes the following key components:

- **ChatGPT-4 API Costs:** The project will utilize the ChatGPT-4 API, which costs $2.50 per 1 million input tokens and $10.00 per 1 million output tokens. Based on an estimated usage of 10 million tokens (combined input and output), the total cost for API usage is approximately $125.
- **Turnitin API Costs:** For plagiarism detection, the project plans to integrate the Turnitin API. During the testing and development phase, the service will be used with a small group of 10 users, costing $3 per user annually. This results in an annual cost of:

$$10\,\text{users} \times 3\,\text{USD/user} = 30\,\text{USD/year}.$$

Total Estimated Budget: The total estimated budget for testing and development is approximately $155, considering both ChatGPT-4 and Turnitin API costs.

7 Supportive Documents

7.1 Survey

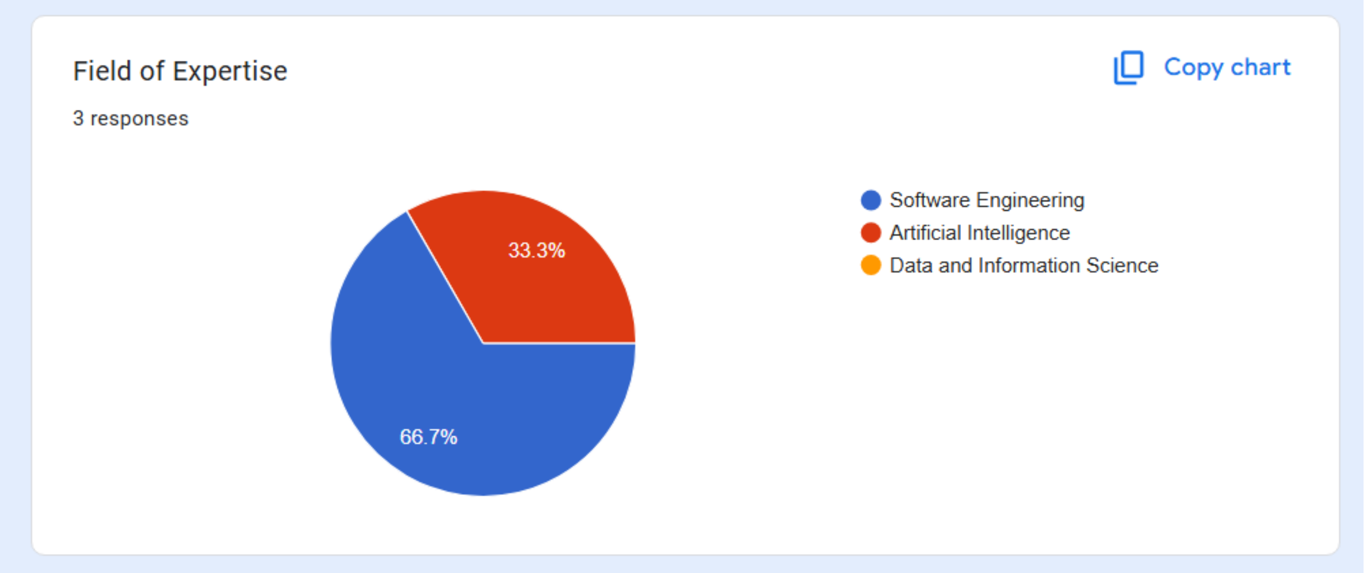


Figure 19: Question 1

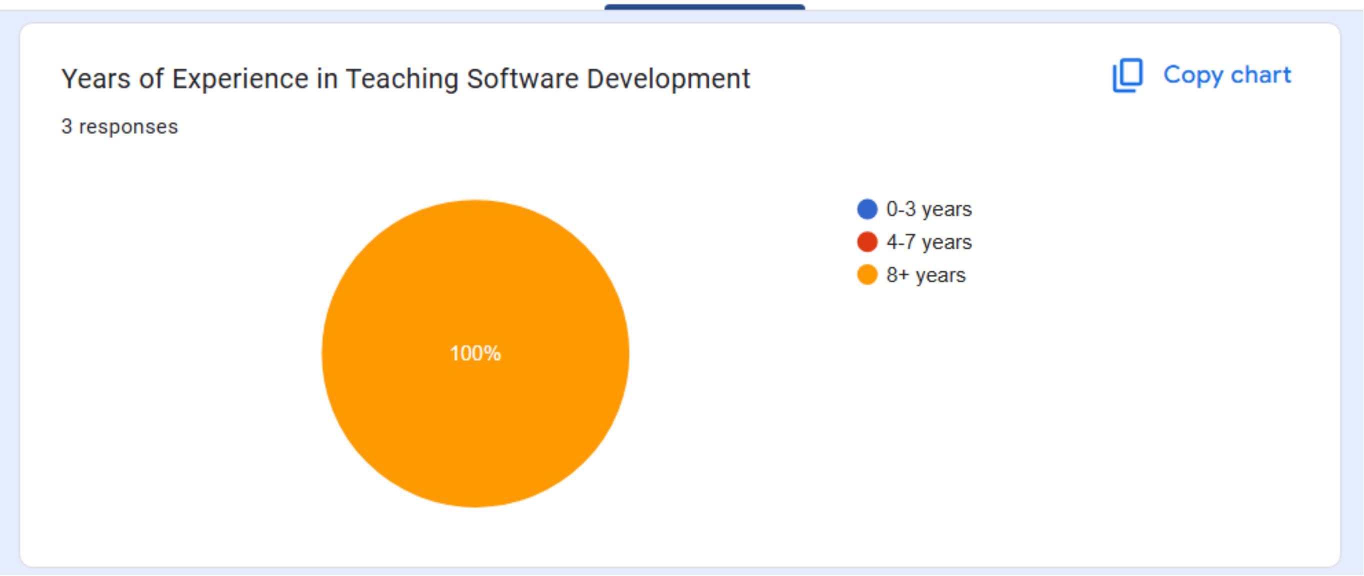
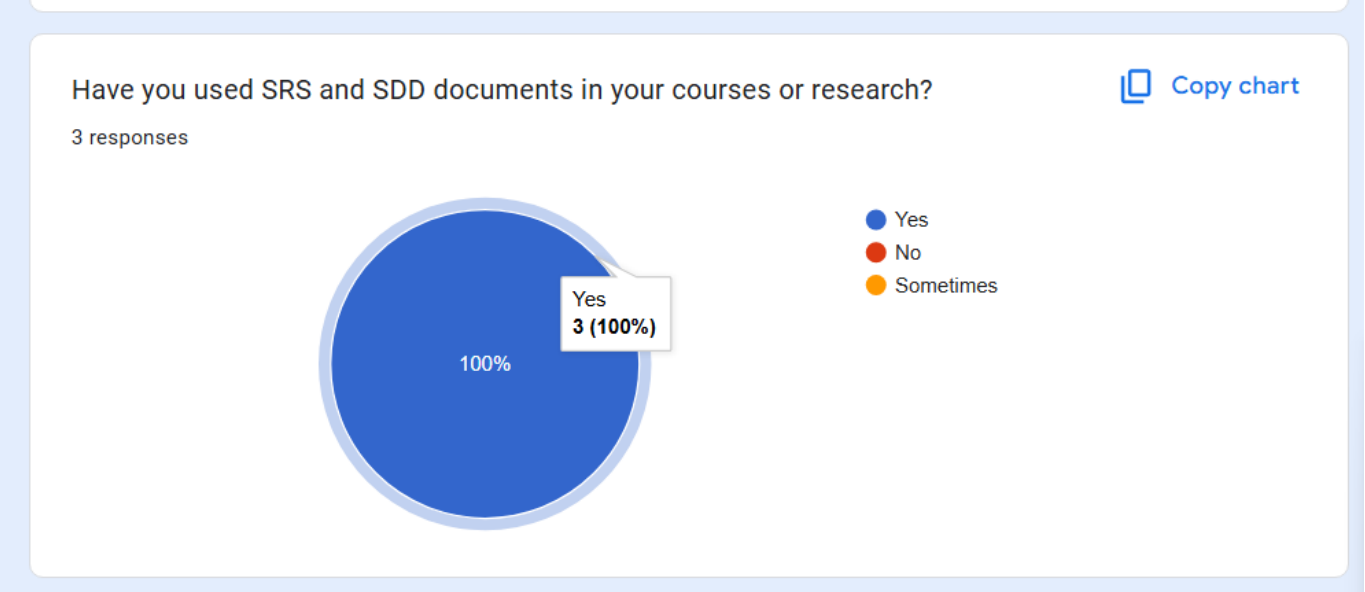


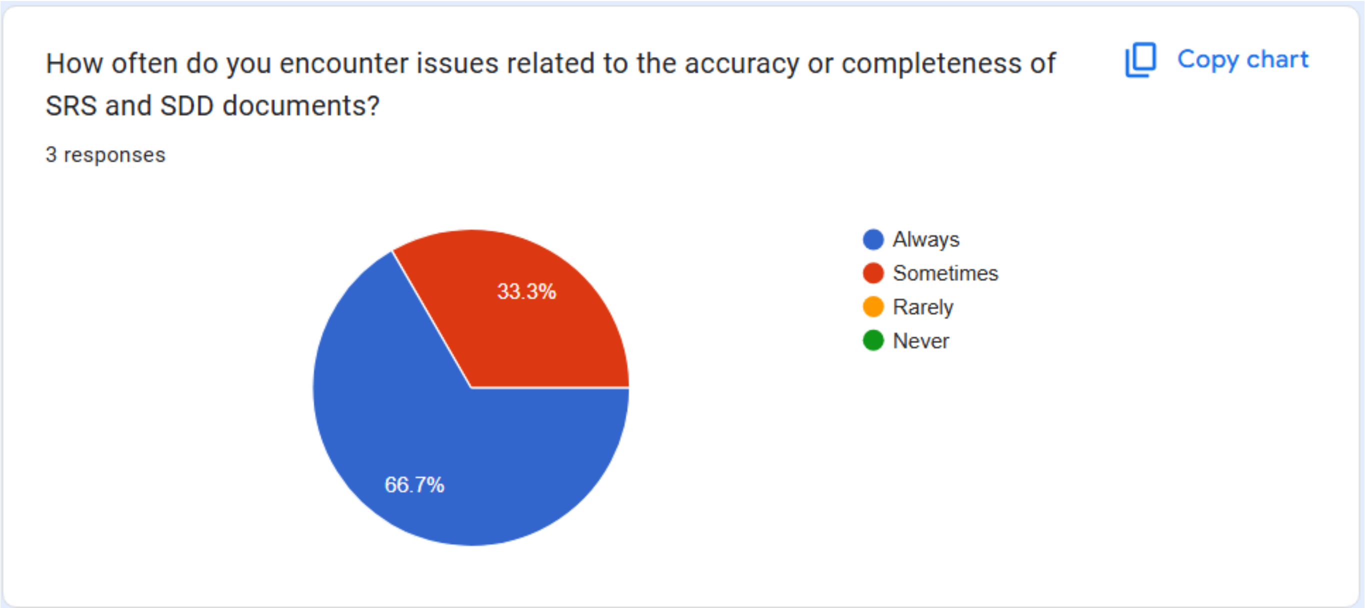
Figure 20: Question 2



Have you used SRS and SDD documents in your courses or research?

3 responses

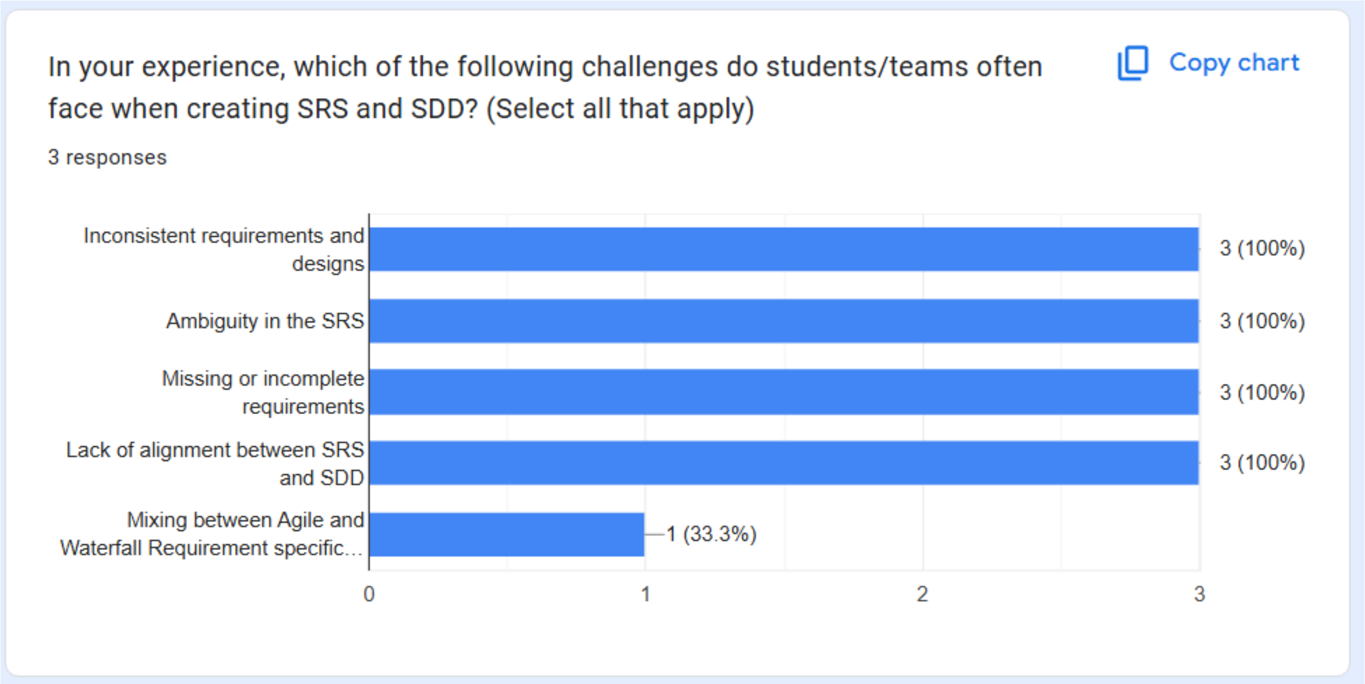
Figure 21: Question 3



How often do you encounter issues related to the accuracy or completeness of SRS and SDD documents?

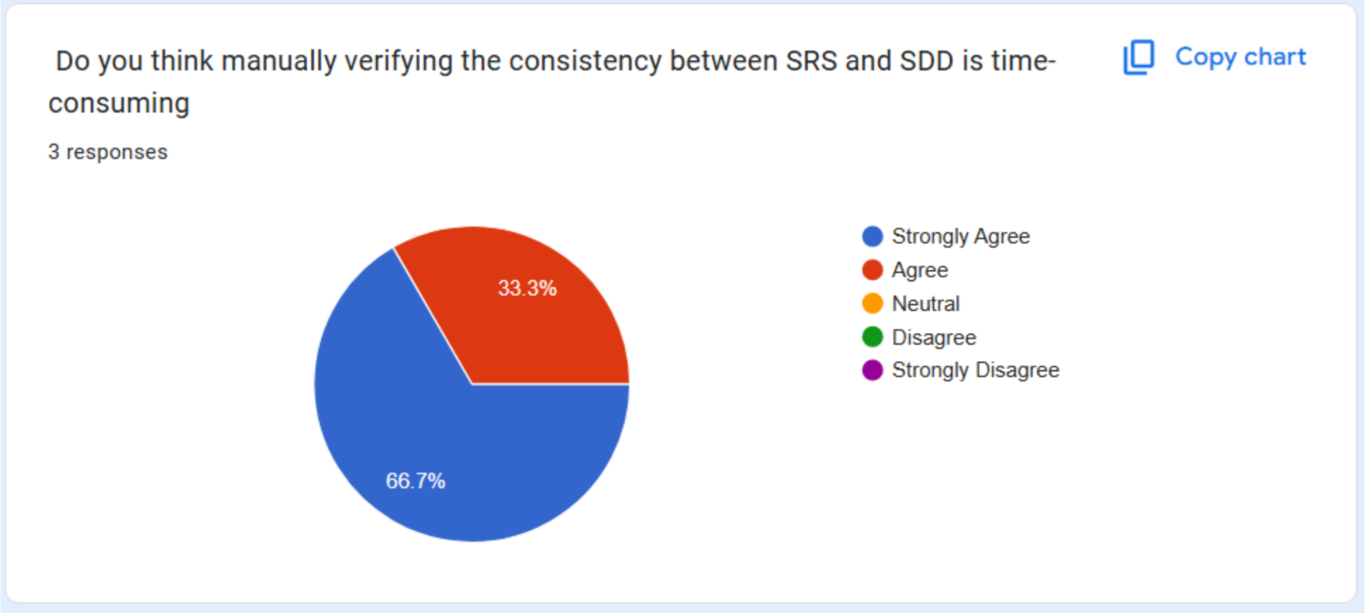
3 responses

Figure 22: Question 4


In your experience, which of the following challenges do students/teams often face when creating SRS and SDD? (Select all that apply)

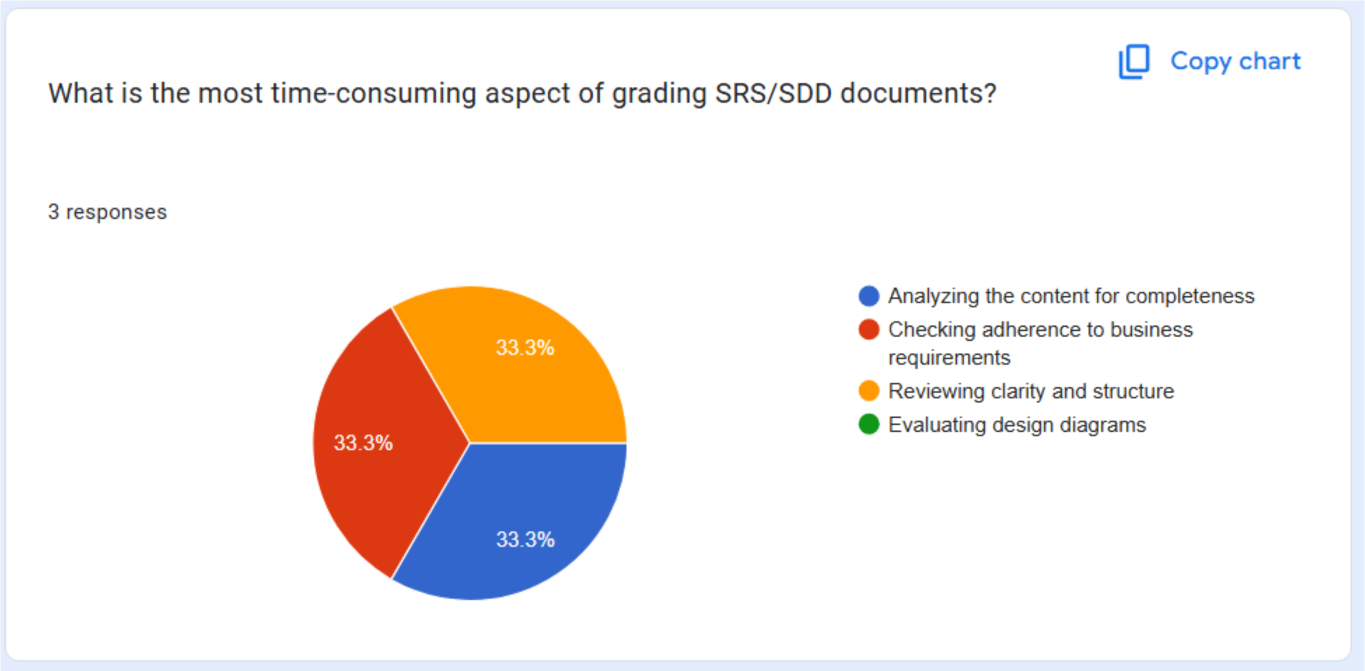
3 responses

Figure 23: Question 5


Do you think manually verifying the consistency between SRS and SDD is time-consuming

3 responses

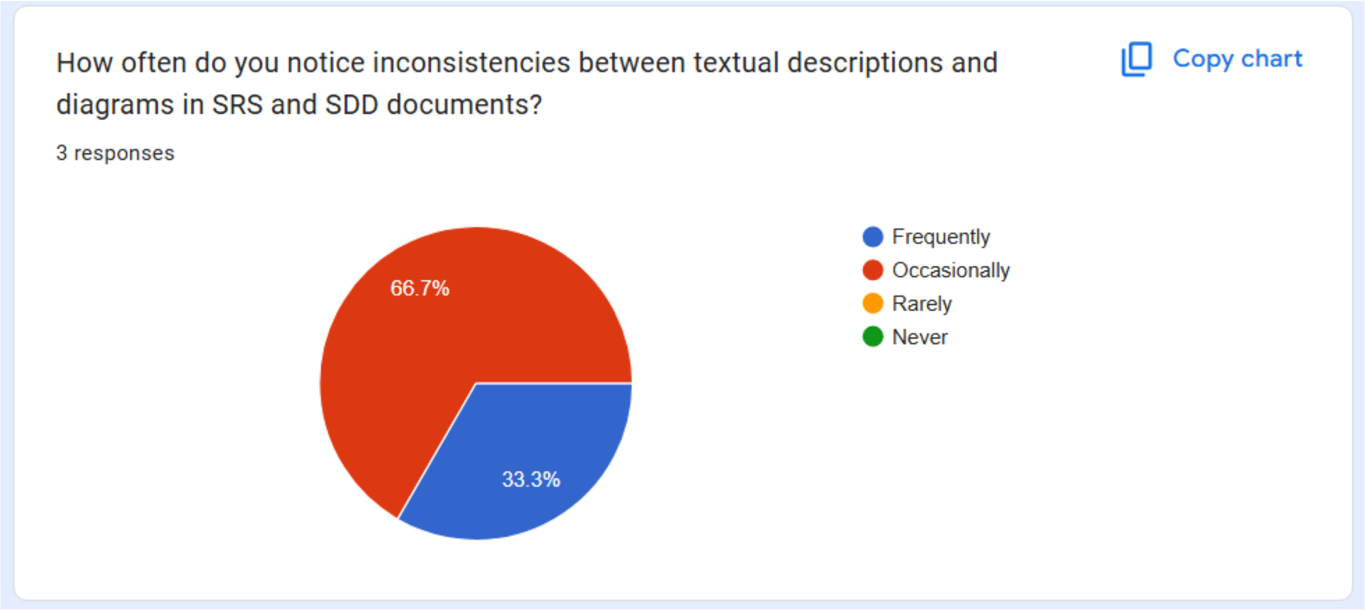
Figure 24: Question 6



What is the most time-consuming aspect of grading SRS/SDD documents?

3 responses

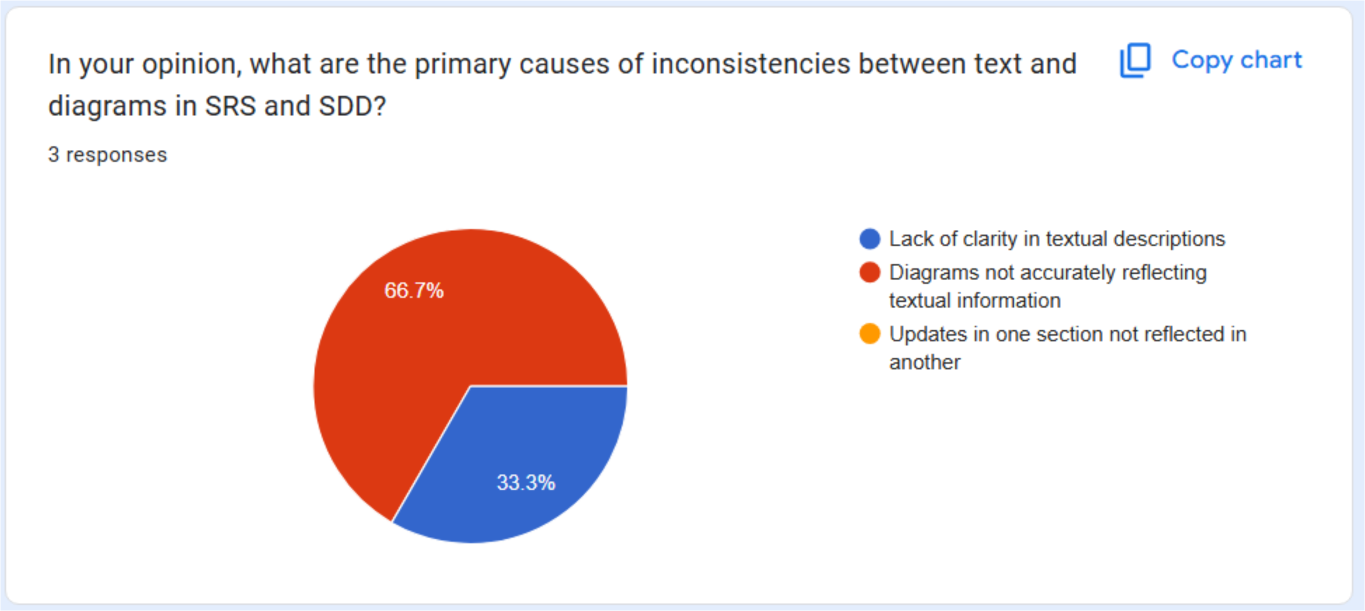
Figure 25: Question 7



How often do you notice inconsistencies between textual descriptions and diagrams in SRS and SDD documents?

3 responses

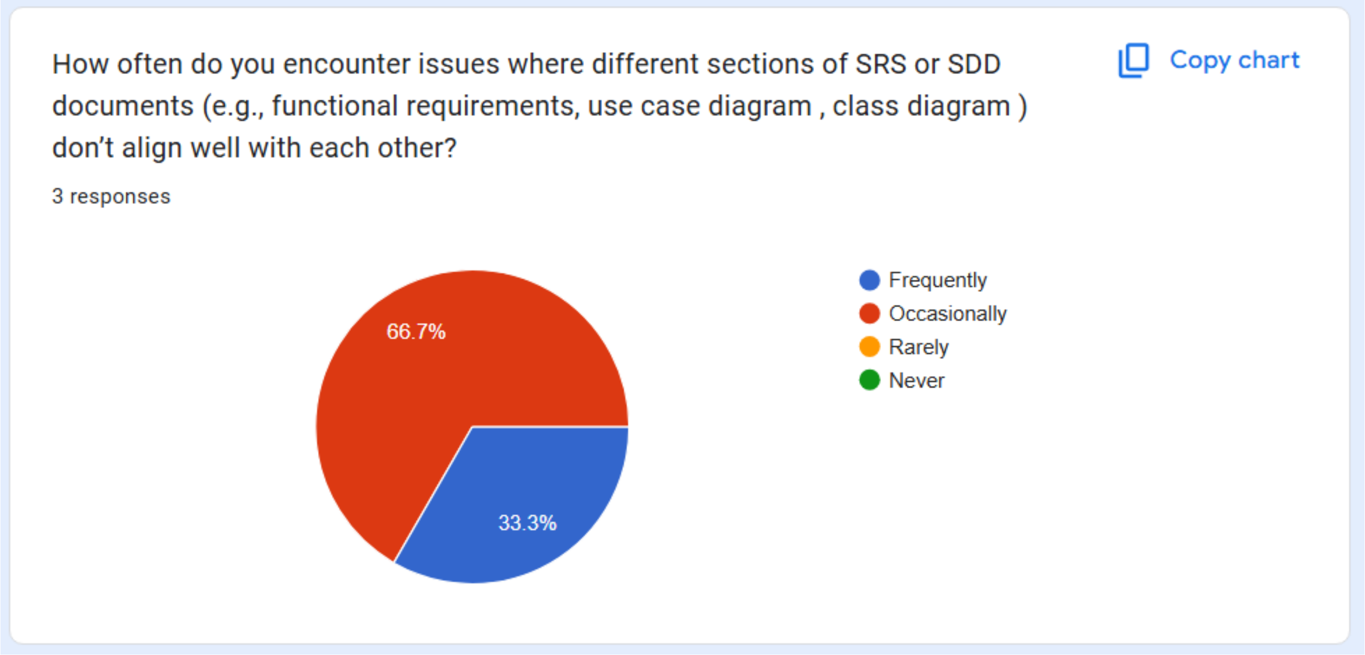
Figure 26: Question 8



In your opinion, what are the primary causes of inconsistencies between text and diagrams in SRS and SDD?

3 responses

Figure 27: Question 9



How often do you encounter issues where different sections of SRS or SDD documents (e.g., functional requirements, use case diagram , class diagram) don’t align well with each other?

3 responses

Figure 28: Question 10

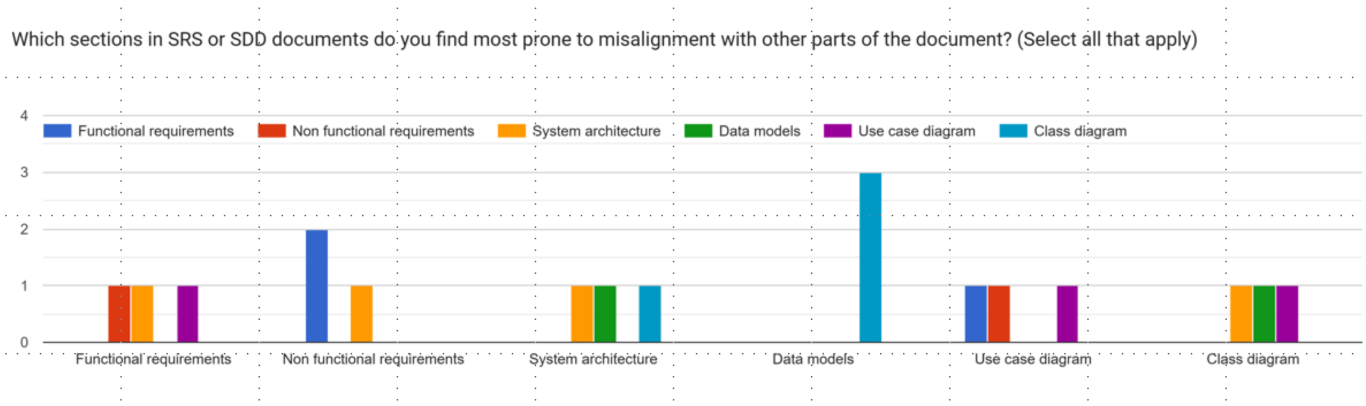
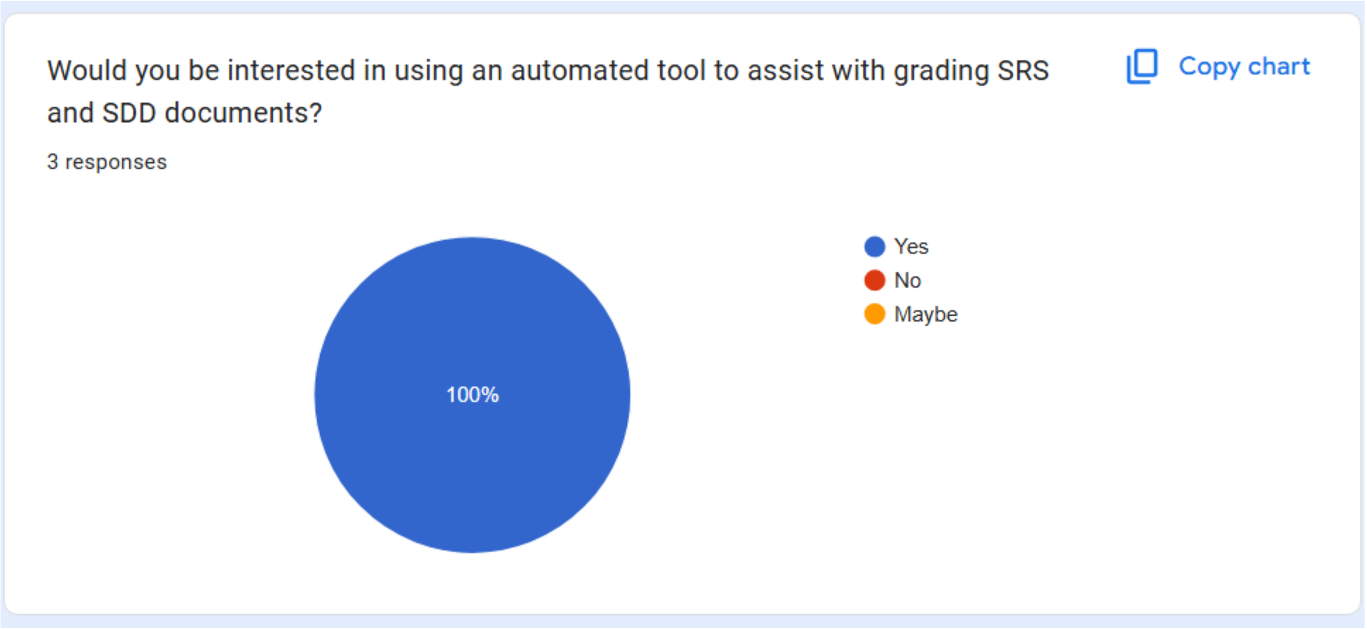


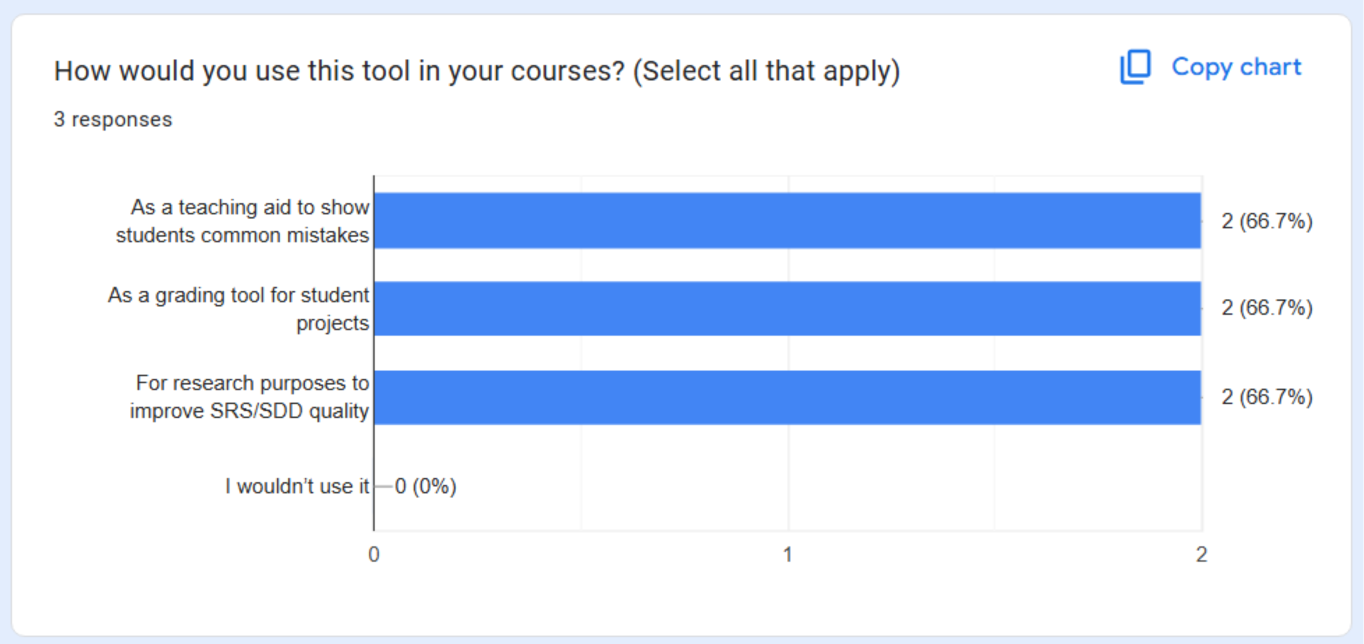
Figure 29: Question 11



Would you be interested in using an automated tool to assist with grading SRS and SDD documents?

3 responses

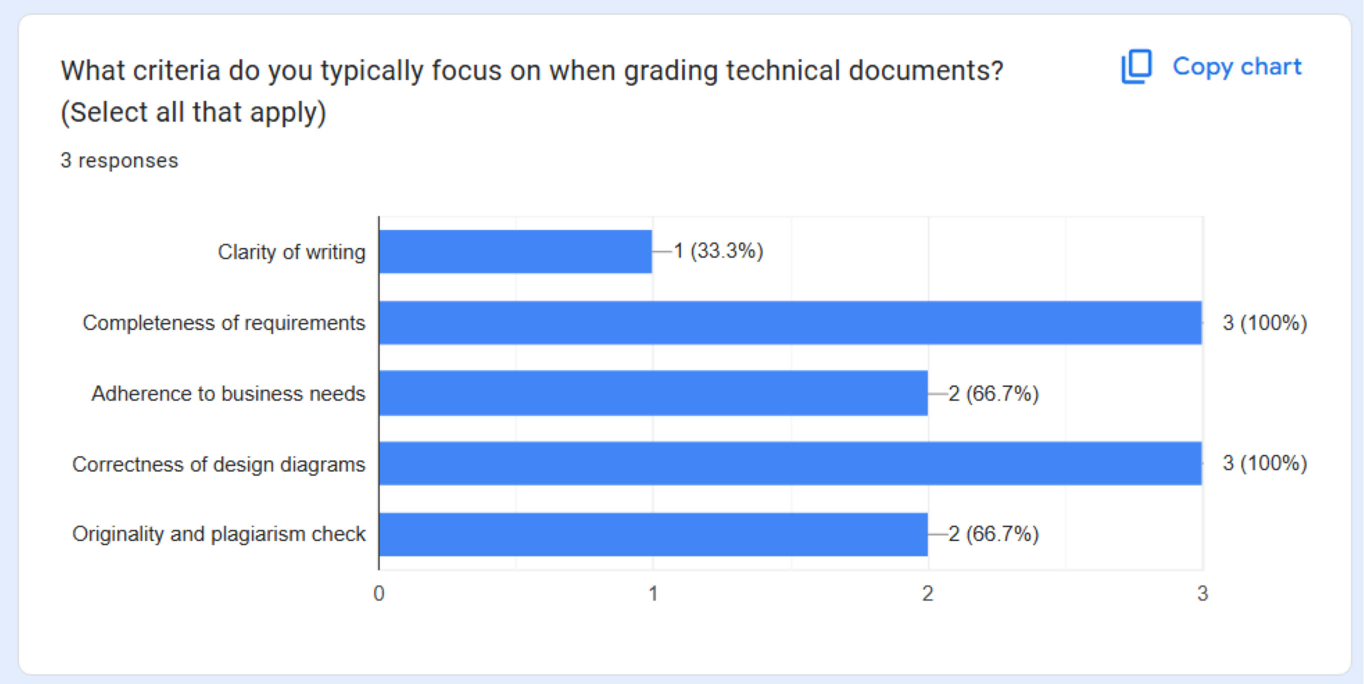
Figure 30: Question 12



How would you use this tool in your courses? (Select all that apply)

3 responses

Figure 31: Question 13



What criteria do you typically focus on when grading technical documents? (Select all that apply)

3 responses

Figure 32: Question 14

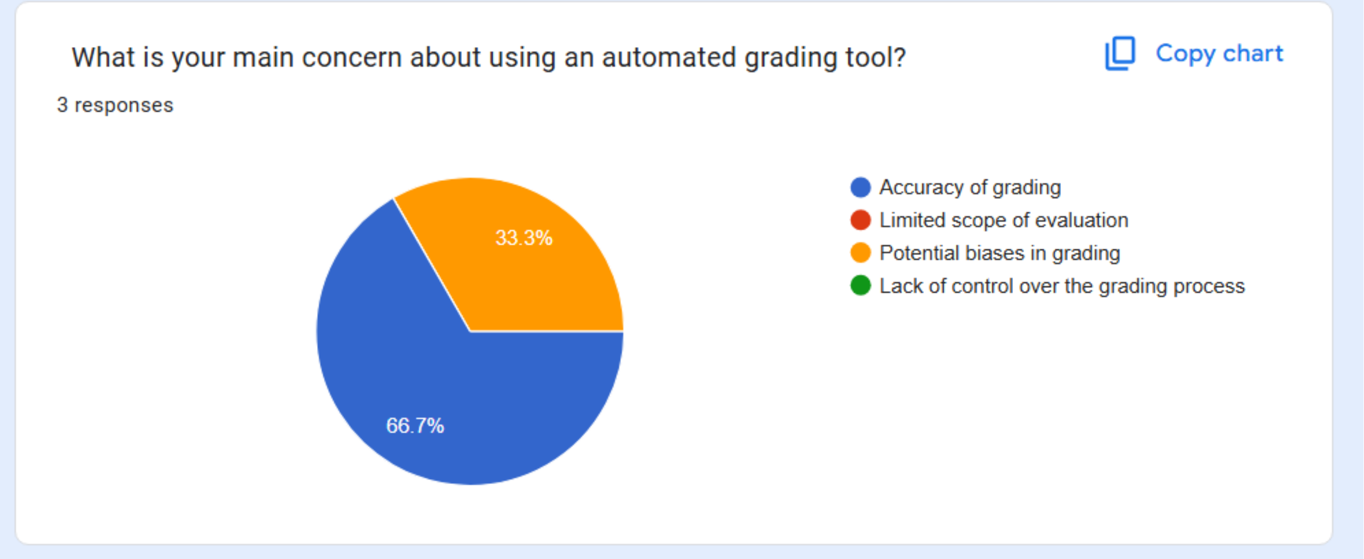


Figure 34: Question 16

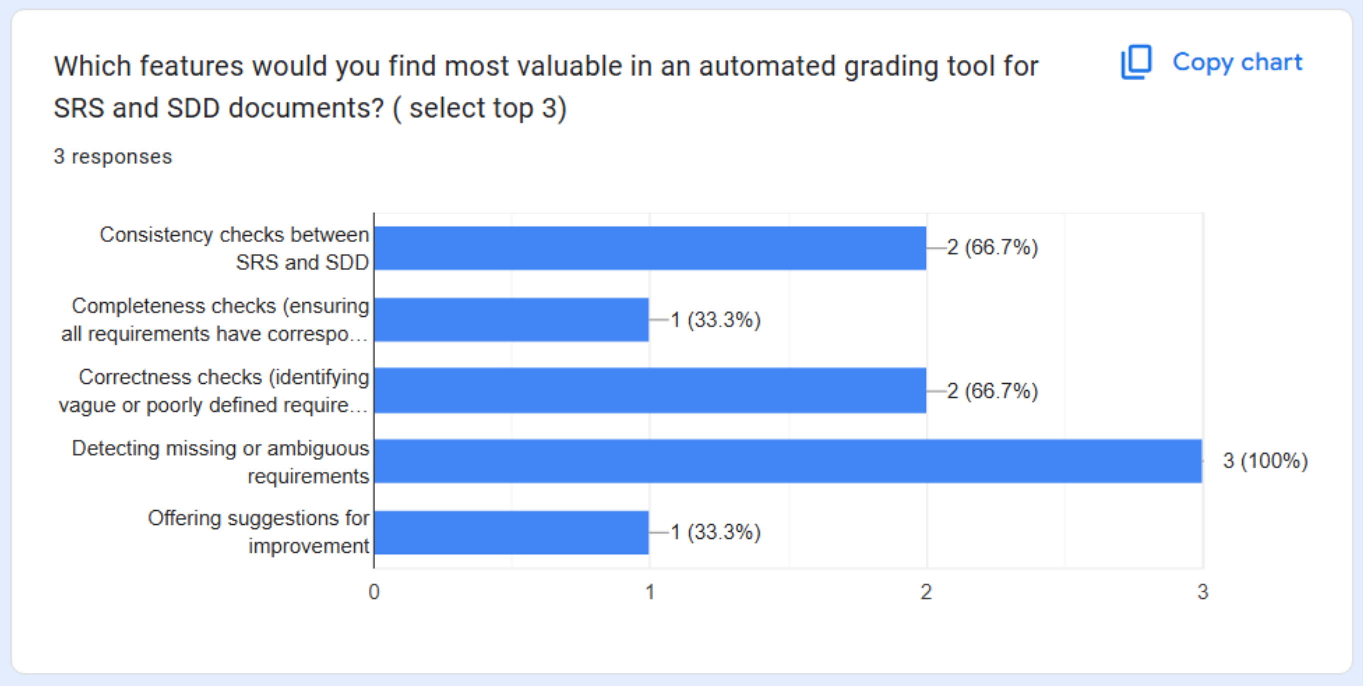


Figure 33: Question 15

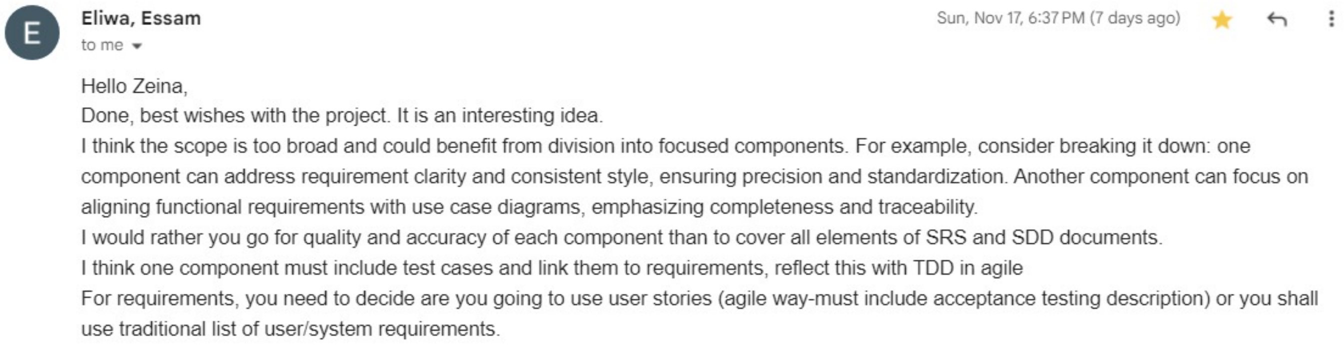
7.2 Dataset

The dataset used in this project spans the last three years and contains grading data for SRS and SDD documents from multiple teams. The dataset consists of the following records:

- **2020**: 27 teams
- **2022**: 27 teams
- **2023**: 31 teams

This data serves as the foundation for evaluating the performance of the system in grading and analyzing SRS and SDD documents. It helps in training and refining the tool to ensure accuracy and relevance across different years and teams.

7.3 Contact



Eliwa, Essam

to me

Sun, Nov 17, 6:37 PM (7 days ago)

Hello Zeina,

Done, best wishes with the project. It is an interesting idea.

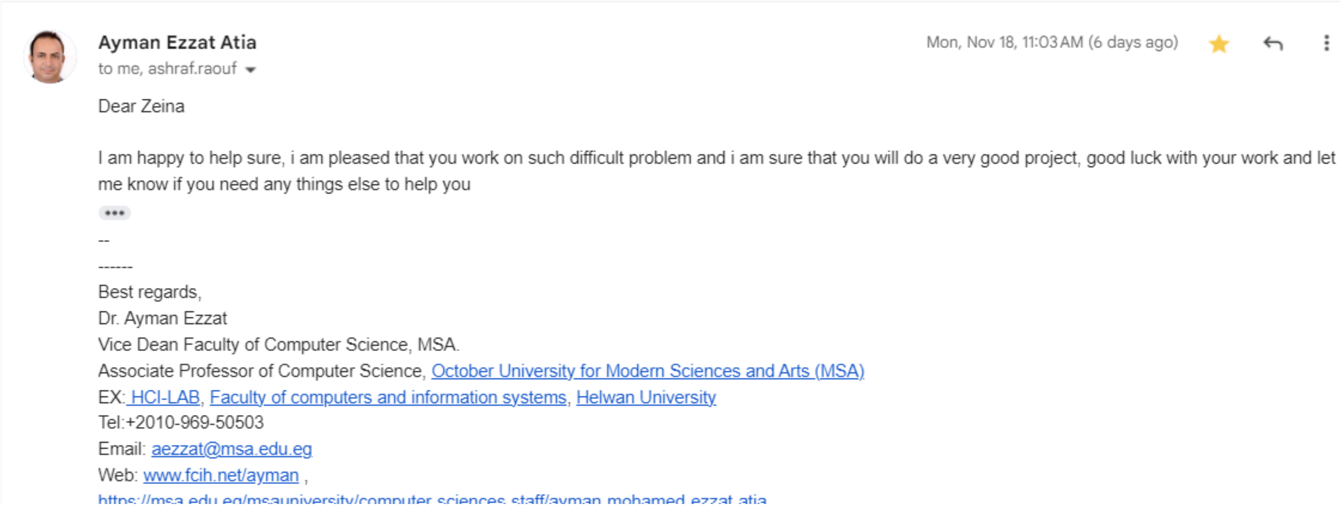
I think the scope is too broad and could benefit from division into focused components. For example, consider breaking it down: one component can address requirement clarity and consistent style, ensuring precision and standardization. Another component can focus on aligning functional requirements with use case diagrams, emphasizing completeness and traceability.

I would rather you go for quality and accuracy of each component than to cover all elements of SRS and SDD documents.

I think one component must include test cases and link them to requirements, reflect this with TDD in agile

For requirements, you need to decide are you going to use user stories (agile way-must include acceptance testing description) or you shall use traditional list of user/system requirements.

Figure 35: Dr Essam Eliawa Suggestion Email



Ayman Ezzat Atia

to me, ashraf.raouf

Mon, Nov 18, 11:03 AM (6 days ago)

Dear Zeina

I am happy to help sure, i am pleased that you work on such difficult problem and i am sure that you will do a very good project, good luck with your work and let me know if you need any things else to help you

--

Best regards,
Dr. Ayman Ezzat
Vice Dean Faculty of Computer Science, MSA.
Associate Professor of Computer Science, October University for Modern Sciences and Arts (MSA)
EX: HCI-LAB, Faculty of computers and information systems, Helwan University
Tel:+2010-969-50503
Email: aezzat@msa.edu.eg
Web: www.fcih.net/ayman ,
https://msa.edu.eg/msauniversity/computer sciences staff/ayman mohamed ezzat atia

Figure 36: Dr Ayman Ezzat Email

References

[1] Yucheng Chu et al. "A LLM-Powered Automatic Grading Framework with Human-Level Guidelines Optimization". In: *arXiv preprint arXiv:2410.02165* (2024).

[2] Xiaoming Xi. *Automated scoring and feedback systems: Where are we and where are we heading?* 2010.

[3] Julio C Caiza and Jose M Del Alamo. "Programming assignments automatic grading: review of tools and implementations". In: *INTED2013 Proceedings* (2013), pp. 5691–5700.

[4] Amira A Alshazly, Ahmed M Elfatatry, and Mohamed S Abougabal. "Detecting defects in software requirements specification". In: *Alexandria Engineering Journal* 53.3 (2014), pp. 513–527.

[5] Rahul Kr Chauhan et al. "Automated content grading using machine learning". In: *arXiv preprint arXiv:2004.04300* (2020).

[6] Syed Waqas Ali, Qazi Arbab Ahmed, and Imran Shafi. "Process to enhance the quality of software requirement specification document". In: *2018 International Conference on Engineering and Emerging Technologies (ICEET)*. IEEE. 2018, pp. 1–7.

[7] Maninder Singh. "Automated validation of requirement : a machine learning approach". In: *2018 IEEE 26th International Requirements Engineering Conference (RE)*. IEEE. 2018, pp. 460–465.

[8] O Fagbohun et al. "Beyond traditional assessment: Exploring the impact of large language models on grading practices". In: *Journal of Artifical Intelligence and Machine Learning & Data Science* 2.1 (2024), pp. 1–8.

[9] Marcus Messer et al. "Automated Grading; Feedback; Assessment; Computer Science Education; Systematic Literature Review". In: (2023).

[10] Gloria Ashiya Katuka, Alexander Gain, and Yen-Yun Yu. "Investigating Automatic Scoring and Feedback using Large Language Models". In: *arXiv preprint arXiv:2405.00602* (2024).

[11] Humasak Simanjuntak. "Proposed framework for automatic grading system of ER diagram". In: *2015 7th International Conference on Information Technology and Electrical Engineering (ICITEE)*. IEEE. 2015, pp. 141–146.

[12] iParadigms LLC. *Turnitin*. Last visited on 14/11/2023. 1998. URL: `https://www.turnitin.com/`.