

UNIVERSIDADE ABERTA

INSTITUTO SUPERIOR TÉCNICO



## TÍTULO DA TESE

An approach in Software Requirements Specification with Generative AI and RSL framework

### Nome completo do candidato

Paulo Alexandre Fonseca de Campos Rêgo

### Mestrado em Informação e Sistemas Empresariais

(mestrado em associação)

Tese orientada pelo (a) Professor (a) Doutor (a) Nome completo

Professor Doutor Miguel Leitão Bignolas Mira da Silva

Professor Doutor Alberto Manuel Rodrigues da Silva

Ano de conclusão

2025

## **Abstract**

This study explores the impact of Artificial Intelligence (AI) in addressing the inherent ambiguities and inefficiencies of natural language processing within the domain of software requirements specification (SRS). By coupling the capabilities of the ITLingo RSL language and integrating them with artificial intelligence tools, the research aims to streamline and enhance the process of deriving accurate software engineering requirements from natural language inputs. The approach proposes an exploration in requirements engineering, resourcing to generative AI technologies to bridge the gap between informal natural language (NL) requirements and their rigorous specifications in RSL and back to natural language from RSL specifications. This approach aims to improve the accuracy of software engineering practices and position AI as a complementary tool in requirements engineering. Through the development and evaluation of an AI-driven artifact, the “GenAI4RSL” chatbot, this study aims to demonstrate the feasibility and benefits of automating SRS generation, providing insights into the future of AI-assisted software development processes. The evaluation of the "GenAI4RSL" chatbot demonstrated its potential to improve the SRS process by automating the generation of structured requirements. Users found the system intuitive, though some challenges were identified, particularly in handling more complex requirements. Accuracy in natural language processing remains an area for improvement. Despite these limitations, participants appreciated the tool's ability to reduce the time spent on repetitive tasks.

In conclusion, while the GenAI4RSL chatbot shows promise in streamlining requirements engineering, further refinements are needed to improve accuracy and adapt it to the specific needs of different user roles. These improvements will be essential for broader applications in the software development field.

**Keywords:** Software Requirements Specification, ITLingo, RSL (Requirements Specification Language), Natural Language, Generative Artificial Intelligence, Chatbot.

## Resumo

Este estudo explora o impacto da Inteligência Artificial (IA) na resolução das ambiguidades e ineficiências inerentes ao processamento da língua natural no domínio da especificação de requisitos de software (SRS). Ao aliar as capacidades da linguagem ITLingo RSL e integrá-las com ferramentas de inteligência artificial, a pesquisa visa otimizar e aprimorar o processo de extração de requisitos de engenharia de software que sejam precisos, a partir de entradas em língua natural. A abordagem propõe uma exploração na engenharia de requisitos, utilizando tecnologias de IA generativa para colmatar a lacuna entre os requisitos informais em língua natural e as suas especificações rigorosas em RSL, e vice-versa, de RSL para a língua natural. Esta abordagem visa melhorar significativamente a precisão das práticas de engenharia de software, posicionando a IA como uma ferramenta complementar nas metodologias de engenharia de requisitos. Através do desenvolvimento e avaliação de um artefacto impulsionado por IA, este estudo pretende demonstrar a viabilidade e os benefícios da automatização da geração de SRS, oferecendo uma visão sobre o futuro dos processos de desenvolvimento de software assistidos por IA. A avaliação do chatbot "GenAI4RSL" demonstrou o seu potencial para melhorar o processo de SRS através da automação da geração de requisitos estruturados. Os utilizadores consideraram o sistema intuitivo, embora tenham sido identificados alguns desafios, em particular na gestão de requisitos mais complexos. A precisão no processamento de língua natural permanece uma área a ser aprimorada. Apesar destas limitações, os participantes valorizaram a capacidade da ferramenta em reduzir o tempo despendido em tarefas repetitivas.

Em conclusão, embora o "GenAI4RSL" mostre potencial para otimizar a engenharia de requisitos, são necessárias melhorias adicionais para aumentar a precisão e adaptá-lo às necessidades específicas de diferentes perfis de utilizadores. Estas melhorias serão essenciais para uma aplicação mais ampla no campo do desenvolvimento de software.

**Palavras-chave:** Especificação de Requisitos de Software, RSL (Linguagem de Especificação de Requisitos), ITLingo, Linguagem Natural, Inteligência Artificial Generativa, Chatbot.

## **Acknowledgments**

I extend my heartfelt gratitude to my thesis supervisors, Prof. Alberto Silva and Prof. Miguel Mira da Silva, for their invaluable guidance, encouragement, and generous knowledge sharing, all of which were essential in making this research possible.

Finally, I am immensely grateful to the participants in the interviews, as well as to my friends and colleagues, who have consistently supported me and contributed to my personal growth. To each of you—thank you.

# Table of Contents

<b>1</b>	<b>INTRODUCTION.....</b>	<b>3</b>
1.1	CONTEXT AND MOTIVATION.....	4
1.2	PROBLEM ADDRESSED.....	5
1.3	RESEARCH GOALS.....	5
1.4	RESEARCH METHODOLOGY.....	6
1.5	DOCUMENT STRUCTURE.....	9
<b>2</b>	<b>BACKGROUND.....</b>	<b>11</b>
2.1	SOFTWARE REQUIREMENTS SPECIFICATION.....	11
2.2	GENERATIVE ARTIFICIAL INTELLIGENCE.....	11
2.3	THE ITLINGO RSL LANGUAGE.....	15
2.4	ARTIFICIAL INTELLIGENCE SUPPORTING SOFTWARE REQUIREMENTS SPECIFICATION.....	19
<b>3</b>	<b>LITERATURE REVIEW.....</b>	<b>22</b>
3.1	PLANNING.....	22
3.1.1	Research Question.....	23
3.1.2	Study Selection Criteria.....	23
3.1.3	Literature Search Strategy.....	24
3.2	CONDUCTING.....	25
3.3	REPORTING.....	27
3.4	DISCUSSION.....	38
<b>4</b>	<b>THE GENAI4RSL CHATBOT.....</b>	<b>42</b>
4.1	ARTIFICIAL INTELLIGENCE CHATBOT GENAI4RSL AS A SOLUTION.....	43
4.2	DATA COLLECTION & PRE-PROCESSING.....	45
4.3	INITIAL TRAINING.....	51
4.4	ITERATIVE PROMPT ENGINEERING & FEEDBACK.....	62
<b>5</b>	<b>DEMONSTRATION.....</b>	<b>67</b>

5.1	DEVELOP TEST INSTRUCTIONS .....	72
5.2	TEST EXECUTION .....	72
<b>6</b>	<b>EVALUATION.....</b>	<b>75</b>
6.1	PARTICIPANTS .....	80
6.2	EVALUATION CRITERIA.....	81
6.3	SURVEY ANALYSIS.....	81
6.4	QUANTITATIVE ANALYSIS.....	81
6.4.1	Demographics and Professional Information .....	81
6.4.2	GenAI4RSL Usability and Impact .....	82
6.4.3	Conversion Accuracy and Process Evaluation .....	83
6.4.4	Correlating GenAI4RSL Performance with User Roles .....	84
6.4.5	Average Time Spent by Respondent Role .....	86
6.4.6	Evaluation of Requirements Engineering Practices .....	87
6.5	QUALITATIVE ANALYSIS .....	90
6.6	INTERVIEWS ANALYSIS .....	92
6.6.1	Broader Context of Feedback .....	92
6.6.2	Suggestions for Improvement .....	93
6.6.3	Impact of AI on Specific Roles .....	94
6.6.4	General Perception of AI .....	94
6.7	DISCUSSION OF EVALUATION .....	95
6.7.1	User Interface Enhancements.....	95
6.7.2	Improved Conversion Processes .....	96
6.7.3	Accuracy Enhancements .....	96
6.7.4	Incorporating software project architecture .....	96
6.7.5	Integration with Existing Tools.....	97
6.7.6	Role-Specific Adjustments .....	97

6.7.7	Broader Impact on the Software Development Industry.....	97
<b>7</b>	<b>CONCLUSION.....</b>	<b>100</b>
7.1	SUMMARY OF RESEARCH.....	100
7.2	LIMITATIONS.....	102
7.3	FUTURE WORK.....	104
	<b>REFERENCES .....</b>	<b>106</b>
	<b>APPENDIX A.....</b>	<b>111</b>
	<b>APPENDIX B.....</b>	<b>113</b>
	<b>APPENDIX C.....</b>	<b>119</b>
	<b>APPENDIX D.....</b>	<b>126</b>
	<b>APPENDIX E.....</b>	<b>190</b>

## List of Tables

Table 3.1 - Main concepts related to studies .....	28
Table 3.2 - Type of Requirements and Modeling Languages used by the study.....	29
Table 4.3 – RSL’s Key Constructs .....	48
Table 4.4 - Training Dataset for the GenAI4RSL .....	50
Table 4.5 - Software Requirements Specification Structure in NL .....	63
Table 5.6 - Requirements translation to RSL .....	69
Table 6.7 - Demographic statistics .....	82
Table 6.8 - Usability and Impact Scores .....	82
Table 6.9 - Accuracy and Process Evaluation .....	83
Table 6.10 - Comparing Process Evaluation and Conversion Accuracy with Respondent’s Roles .....	84
Table 6.11 - Average Time Spent by Respondent Role .....	86
Table 6.12 - Summary of Survey Responses on Requirements Engineering Practices .....	87
Table 10.13 - List of Studies included in analysis.....	113
Table 10.14 - Correlation of Technologies with Requirements by Study .....	115
Table 12.15 - Statistical Summary of Quantitative Questions.....	190

## List of Figures

Figure 3.1 - Conducting stages of the review .....	26
Figure 3.2 – Number of studies by year .....	27
Figure 3.3 - Type of publication by studies .....	27
Figure 3.4 – Diagram representing the relationships between Artificial Intelligence, Machine Learning, and Deep Learning (extracted from Bellini et al., 2022, p. 3) [8] .....	34
Figure 4.5 - GenAI4RSL-based workflow .....	45
Figure 5.6 - Application Architecture (Application Usage) .....	69

# List of Acronyms

**AI** (Artificial Intelligence)  
**AIUEF** (AI-augmented Usability Evaluation Framework)  
**AMLCP** (Probabilistic Action-Model Learning and Critical Path algorithm)  
**GEMINI** (Google's Artificial Intelligence-powered chatbot)  
**BiLSTM** (Bidirectional Long Short-Term Memory)  
**CNN** (Convolutional Neural Network)  
**CRF** (Conditional Random Field)  
**DevOps** (Development and Operations)  
**DizReq** (Distributed Intelligent Requirements Engine)  
**DSL** (Domain-Specific Language)  
**DSR** (Design Science Research)  
**DSM** (Domain-Specific Modeling)  
**Ernie** (Enhanced Representation through Knowledge Integration)  
**GenAI4RSL** (Generative Artificial Intelligence for Requirements Specification Language)  
**GORE** (Goal-Oriented Requirements Engineering)  
**GPT** (Generative Pre-Trained Transformer)  
**Grok** (X's Artificial Intelligence chatbot)  
**HCI** (Human-Computer Interaction)  
**IoT** (Internet of Things)  
**IS** (Information System)  
**jUCMNav** (Java User Requirements Notation (URN) Navigator)  
**LLaMA** (Large Language Model Meta Artificial Intelligence)  
**LLM** (Large Language Model)  
**ML** (Machine Learning)  
**MLP** (Multi-layer Perceptron)  
**MDE** (Model-Driven Engineering)  
**NL** (Natural Language)  
**NLP** (Natural Language Processing)  
**OpenCog** (An open-source artificial intelligence framework)  
**POS** (Part-Of-Speech)  
**RE** (Requirement Engineering)  
**RSL** (Requirements Specification Language)  
**SRS** (Software Requirement Specification)

**SLR** (Systematic Literature Review)

**spaCy** (Open-source library for Natural Language Processing in Python)

**SVC** (Support Vector Classifier)

**SVM** (Support Vector Machine)

**UML** (Unified Modeling Language)

**URN** (User Requirements Notation)

# **1**

## **Introduction**



## 1 Introduction

In the evolving landscape of software development, the integration of Artificial Intelligence (AI) in software requirements specification (SRS) is emerging as a crucial area of research and application [61]. While AI technologies have been increasingly applied across various stages of software development, their potential to enhance the accuracy, efficiency, and effectiveness of software requirements specification remains a prolific territory. This study intends to provide a comprehensive analysis of the current state of AI integration in SRS, exploring theoretical frameworks and practical implementations. The focus is to unravel the complexities, benefits, and challenges associated with this integration, offering valuable insights for academic researchers and industry practitioners.

The motivation behind this research stems from the perspective of AI to significantly improve the quality of software requirements specification by reducing human errors, ambiguities, and inconsistencies. This research seeks to explore how AI can contribute by creating more reliable SRS documents, thereby enhancing the overall quality of software projects.

The primary objective of this study is to elucidate the technological underpinnings of AI-based methods and assess their influence on the efficacy and accuracy of requirements specification. By concentrating on project managers and requirement engineering, this research seeks to provide academic relevancy.

In light of the ubiquitous adoption of AI in software development, this report explores critical research questions: What is AI's impact on the reliability, performance, and cost models within the requirements specification process? How do AI-driven methodologies compare with traditional practices in terms of benefits and risks?

The significance of this study transcends theoretical exploration; it aims to provide practical contributions to support professional practices in achieving greater efficiency and fostering innovation. For researchers, the findings aim to bridge the gap between speculative potential and empirical evidence, fostering a deeper understanding of AI's role in software engineering.

The objective of this research is to explore technologies based on artificial intelligence and their connection with software engineering to improve requirements specification accuracy, mitigating ambiguities, uncertainties, and errors.

## **1.1 Context and Motivation**

The challenging nature of requirements specification in software engineering often leads to inefficiencies, errors, and wasted specification time. This scenario highlights a pressing need for an in-depth exploration of methods that can broadly reduce requirements specification (RS) defects, thus enhancing accuracy in the process. One such method is the use of specialized languages like ITLingo RSL (Requirements Specification Language), which offers a structured and formalized approach to defining software requirements. ITLingo RSL helps minimize ambiguities and inconsistencies by providing clear syntax and semantics for requirement statements, thereby improving the overall quality of the specifications.

Additionally, integrating a chatbot powered by artificial intelligence presents an innovative approach to supporting requirements engineering. Such chatbots facilitate interactive communication between stakeholders and the system, enabling real-time clarification and refinement of requirements. This technology enhances the elicitation process by making it more efficient and user-friendly, reducing the likelihood of misunderstandings and omissions. In this context, the chatbot named GenAI4RSL serves as a tool designed to produce accurate and systematized requirements specifications. This outcome is intended to be achieved by integrating ITLingo RSL to organize the semantic statements of requirements and then converting them into structured natural language.

Consequently, the motivation for this study stems from the necessity to systematize the quality of requirements definition, which has significant implications for the software development process. Gaspar (2012) emphasizes the need to apply robust requirements engineering methodologies to ensure completeness and consistency in requirements

specifications, addressing risks of ambiguities and inconsistencies often found in these specifications [22].

This highlights the urgency for innovative solutions, supported by Artificial Intelligence—such as ITLingo RSL and chatbots—to improve accuracy in requirements engineering. By using and enhancing this solution, this study could lead to more reliable software products and reduced development costs.

## **1.2 Problem Addressed**

This research aims to explore the potential of AI techniques to enhance the accuracy of SRS. The focus is on utilizing AI to interpret, analyze, and generate SRS documents that are coherent, comprehensive, and unambiguous, ensuring accuracy, precision, and efficiency in the requirements specification process. Despite the critical role of SRS in software development, the reliance on manual processes for generating these documents often results in ambiguities, uncertainties, errors, and limitations, adversely affecting the quality and efficiency of the software development lifecycle [22]. By integrating AI with ITLingo RSL, this study seeks to address these challenges by offering a methodological framework for automatically generating SRS that aligns with the principles of natural language requirements. This framework aims to reduce human error and improve the clarity of SRS for both developers and stakeholders, thereby streamlining the software development process.

## **1.3 Research Goals**

The project's research goals are to conduct a comprehensive analysis of the current state-of-the-art use of artificial intelligence tools in software requirements specification and their impact on efficiency and effectiveness. The proposition is to develop an artifact to translate natural language requirements into the ITLingo RSL (Requirements Specification Language) focusing on accuracy to enhance user and software engineer satisfaction. This endeavor includes a detailed exploration of challenges in current specification practices, employing AI and NLP technologies and RSL for the chatbot development, and establishing

evaluation metrics for accuracy. The project aims to significantly impact stakeholders by improving software development processes and advancing the field.

This research proposes the consideration of the following aspects: through an LLM based on GPT-4 and using a user interface like ChatGPT, construct an assistant, also known as a chatbot or conversational agent, that is a type of artificial intelligence (AI) software that can simulate a conversation (or a chat) with a user in natural language. This artifact will be customized with no code development, but throughout prompt engineering. The goal is to use a linguistic pattern called RSL language; specification inputs given to the chatbot can be converted into structured and systematic requirements in natural language.

Therefore, this project intends to achieve the following goals:

RG-1. Develop an AI-powered chatbot, named "GenAI4RSL", by utilizing GPT-4's advanced natural language processing capabilities through prompt engineering and fine-tuning techniques. The objective is to create a system that translates NL requirements specifications into rigorous RSL specifications and back to NL. This process uses the existing ChatGPT framework to offer an intuitive, conversational interface for generating formal software requirements specifications in RSL.

RG-2. Validate the GenAI4RSL within the proposed framework by quantitatively measuring its impact on the accuracy of the generated Requirements Specification Language (RSL) documents compared to traditional manual methods. This will involve conducting comparative analysis and user satisfaction surveys to demonstrate the artifact's relevance and contribution to solving the identified problem.

RG-3. Systematically evaluate the GenAI4RSL accuracy in the generated RSL. This evaluation will employ qualitative and quantitative metrics, aiming to assess the GenAI4RSL impact on streamlining the SRS process, improving user satisfaction, and advancing software development practices.

#### **1.4 Research Methodology**

This study follows the Design Science Research methodology (DSR). The DSR is an iterative methodology that combines principles, practices, and procedures. It provides

guidance for research in Information Systems (IS) as well as other disciplines [26], [45]. Design Science lays emphasis on systematic, testable, and communicable methods [20].

For instance, Hevner et al. propose a set of **guidelines** for the application of DSR in the information systems area, namely including the following aspects [26]:

**Design as an Artifact:** The research shall produce a viable artifact in the form of a construct (e.g., software application or tool), a representation (e.g., new language or extension of a previous notation), a technique (e.g., a process or method), or an instantiation (e.g., a case study that applies such artifacts).

**The relevance of the problem:** The basic objective of DSR is to develop technology-based solutions to relevant and significant business problems.

**The design evaluation:** The quality (e.g., measures in terms of utility or efficacy) of the design artifact shall be demonstrated rigorously through a well-executed evaluation method.

**Research contribution:** Effective DSR shall offer a clear and demonstrable contribution in the area in which the design artifact is applied, such as design foundations and/or design methodologies.

**Research Rigor:** The DSR depends upon rigorous methods applied in both the evaluation and the construction of the design artifact.

**Design as a search process:** The search for an effective artifact depends on the use of the available ways to reach desired outputs while the rules in the problem environment are still satisfied.

**Communicating the results:** The research presentation shall be effective from both the technology and the business perspective.

These guidelines drive the research but can be better translated into the following **phases** [26]:

**Problem identification and motivation (December 2023):** Define the specific research problem, justify the value of a solution, and motivate the researcher to investigate the answer.

**Define the objectives of a solution (January – February 2024):** Infer the objectives of a solution for the defined problem and the knowledge of what is achievable.

Activities:

- Analyze related work involving technologies and methodologies.
- Analyze related work involving related projects.
- Define clear research objectives based on the analysis.
- Write and present the research project report. (January – March 2024)

Design and development (March – April 2024): Creation of GenAI4RSL, an AI-powered chatbot that integrates ITLingo RSL to produce accurate and systematized requirements specifications.

Activities:

1. Data Collection & Pre-processing.
2. Initial Training.
3. Iterative Prompt Engineering & Feedback.

Demonstration (May – June 2024): Demonstrate the efficacy of the GenAI4RSL in translating specifications to ITLingo RSL and back to NL.

Activities:

1. Develop Test Criteria and Execute Tests.
2. Analyze Test Results to assess the GenAI4RSL performance.

Evaluation (June 2024): Examine and measure how well the GenAI4RSL systematizes accurate specifications in ITLingo RSL and NL.

Activities:

1. Conduct an evaluation and time efficiency analysis.
2. Perform a user satisfaction survey to measure the GenAI4RSL impact.

Communication (June – December 2024): Communicate the problem, the GenAI4RSL, and the design, considering its relevance, utility, novelty, and effectiveness to researchers and other relevant audiences.

Activities:

1. Write and defend the MSc dissertation.

Appendix A presents a simple schedule of this project framed according to this DSR methodology. Appendix B presents the list of tables from the literature review reporting.

Appendix C presents the refined requirements specification in RSL and NL from the demonstration phase. Appendix D presents the datasets used to train the GenAI4RSL. Appendix E shows the statistical questionnaire survey results from the evaluation.

## **1.5 Document Structure**

This document is structured in seven chapters, as follows:

Chapter 2 (Background) introduces the main concepts related to Artificial Intelligence and Software Requirements Specification.

Chapter 3 (Literature Review) presents and discusses the state of the art related to the existing literature about the intersection between Artificial Intelligence and Software Requirements Specification.

Chapter 4 (Solution Proposal) describes the solution proposed to establish an improved way to specify requirements by resourcing to artificial intelligence. The demonstration represents the testing of the solution, leading to a refined artifact that efficiently addresses the initially formulated problem.

Chapter 5 (Demonstration) involves validating the functionality of “GenAI4RSL” against a rigorous set of criteria (eg. testing and analyzing its performance using a real-world project).

Chapter 6 (Evaluation) describes the methodology for evaluating the research, including assessing the assistant’s accuracy in translating natural language to RSL, its efficiency, and user satisfaction.

Chapter 7 (Conclusion) presents the main conclusion synthesizing the project's context, motivation, and findings, highlighting AI's role in refining software requirements specification.

Finally, Appendix A presents the planned schedule of this project, as inspired by the DSR methodology. Appendix B presents the list of tables from the literature review reporting. Appendix C displays the prompts used to train the GenAI4RSL. Appendix D presents the datasets used to train the GenAI4RSL. Appendix E shows the statistical questionnaire survey results from the evaluation.

# 2

## **Background**

## **2 Background**

This chapter introduces general concepts and technologies underlying this research, namely: Software Requirements Specification, Artificial Intelligence, Requirements Specification Language (RSL), and the intersection of Artificial Intelligence with Software Requirements Specification.

### **2.1 Software Requirements Specification**

A SRS is a comprehensive description of the intended purpose and environment for software under development. It serves as a critical communication tool among stakeholders, including clients, developers, and project managers, to ensure that all parties have a clear understanding of the software's functionalities, constraints, and operational environment.

This research aims to evaluate the current state of AI technologies and their impact on software engineering, specifically in requirements specification. Through a critical analysis of current academic and industry perspectives, this study aims to demonstrate how combining AI with the RSL language can systematize outcomes and establish a standardized framework for software engineering.

Historically, the concept of SRS emerged alongside the software engineering discipline in the late 1960s, as part of an effort to bring more rigor and structure to software development [44]. The increasing complexity of software systems created the need for a disciplined approach to capture and clarify project specifications, mitigate misunderstandings, and control project growth effectively [51]. Over the years, SRS has evolved, incorporating various methodologies and tools to improve its precision and reliability. However, despite advancements, some challenges persist, including dealing with ambiguous requirements, ensuring stakeholder consensus, and adapting to changes over the software lifecycle [23].

### **2.2 Generative Artificial Intelligence**

Artificial Intelligence (AI) is a branch of computer science focused on creating systems capable of performing tasks that typically require human intelligence. These tasks include

learning from experience, understanding natural language, recognizing patterns, solving problems, and making decisions. AI encompasses a wide range of technologies and approaches, including machine learning, neural networks, natural language processing, computer vision, and robotics. By accessing these technologies, AI systems can process vast amounts of data, identify patterns, and make decisions with minimal human intervention, thereby enabling automation and enhancing the efficiency of various processes across different domains.

Generative AI, a subfield of AI, involves models that can generate new content, such as text, images, or even code, based on the data they have been trained on. These models, like GPT-4, have the ability to produce human-like text, making them valuable tools in various applications, including software engineering [8].

AI's use in software engineering is marked by its transformative potential to disrupt traditional methodologies. The technology has become a differentiator across various domains within the field, offering novel approaches to solving complex problems. Innovations in AI have led to smarter algorithms for code optimization, enhanced predictive models for software testing, and advanced data analytics for informed decision-making [34]. These advancements point towards a future where AI's integration into software engineering practices becomes increasingly seamless and ubiquitous.

Artificial intelligence (AI) chat models represent a significant advancement in natural language processing (NLP). These models simulate human-like conversations and perform diverse tasks, such as answering questions, assisting with creative work, or enhancing productivity in enterprise applications. Prominent examples include ChatGPT, Gemini, Claude, and Microsoft's Copilot. Microsoft Copilot, powered by OpenAI's GPT models, is a productivity-focused AI integrated into Microsoft 365 applications. While it shares conversational capabilities with general-purpose chat models, its primary strength lies in supporting tasks like document editing, data analysis, and meeting summaries, all within enterprise ecosystems.

Each of these systems has unique strengths. For example, Google's Gemini emphasizes integration with vast data sources, including real-time web access, while Anthropic's Claude

is tuned for safety and ethical considerations in AI responses. However, OpenAI's ChatGPT has emerged as a particularly versatile and widely adopted model, available in multiple versions that cater to diverse user needs.

For the purpose of this work, ChatGPT, developed by OpenAI, is an advanced conversational agent based on the Generative Pretrained Transformer (GPT) architecture. It is part of the GPT family of language models, which are trained on vast text datasets to predict and generate language in a contextually accurate manner. ChatGPT builds on this foundation to provide a user-friendly conversational interface for various applications, including customer support, education, programming, and research.

The key features of ChatGPT are:

- **Natural Language Understanding:** ChatGPT excels at understanding context, intent, and subtleties in language, allowing it to produce responses that feel intuitive and human-like.
- **Wide Knowledge Base:** Trained on an extensive dataset covering a wide range of topics, ChatGPT can provide information on general knowledge, technical queries, and creative prompts. However, its knowledge is limited to data up to a certain cutoff (e.g., September 2021 for GPT-4, with newer models featuring periodic updates or live web access in some cases).
- **Multi-Turn Conversations:** ChatGPT is skilled at maintaining context across multiple exchanges, enabling coherent and contextually aware interactions over extended dialogues.
- **Customizability:** Users can guide responses through specific prompts or integrate the model into tailored applications via OpenAI's API.

ChatGPT is available in several versions, each with distinct capabilities:

- **GPT-3.5:** A robust iteration of the GPT-3 model, GPT-3.5 improves response coherence, relevance, and natural tone. It is highly efficient for most conversational tasks.
- **GPT-4:** This version represents a significant leap in AI capabilities. With improved contextual understanding, reasoning, and nuanced response generation, it is

particularly suitable for complex tasks like academic research, detailed technical explanations, and creative writing.

- **GPT-4 Turbo:** This variant offers the advanced capabilities of GPT-4 with increased efficiency and reduced latency, optimized for real-time applications.
- **ChatGPT Enterprise:** Aimed at businesses, this version provides enhanced privacy, security, and API customization options. It is designed to support enterprise-scale deployments with features such as extended context windows and advanced analytics.
- **Code Interpreter (Advanced Data Analysis):** An additional feature available in premium versions, the code interpreter allows ChatGPT to perform advanced calculations, analyze datasets, and generate insights, making it invaluable for data-driven fields.
- **Custom GPTs:** OpenAI also offers tools for users to create personalized versions of ChatGPT tailored to specific domains or requirements, enhancing adaptability across industries.

The training of AI-model tools like GPT-4 involves designing prompts, which essentially serve as a way to “program” the model. By providing specific instructions or examples, users guide the model to complete tasks successfully. These models are versatile and can be applied to a wide range of domains and activities, including content or code generation, summarization, conversational tasks, creative writing, and more.

The OpenAI model GPT-4 released the functionality to users in creating chatbots or assistants, called GPTs. GPTs are custom versions of ChatGPT that users can tailor for specific tasks or topics by combining instructions, knowledge, and capabilities. These GPTs are created and managed inside ChatGPT (OpenAI User Interface) and do not require development knowledge or skills.

Below is an explanation of the primary methods used for developing custom GPTs:

1. **Prompt Engineering:** Involves crafting precise input prompts that guide the model to generate desired responses. This is the simplest form of customization, relying

on the model's existing capabilities rather than altering its architecture or training process.

2. Iterative prompt engineering: is the process of systematically refining prompts to enhance the performance of AI models. This involves repeatedly modifying and testing prompts based on the outputs received, allowing for continuous improvement in the model's responses.
3. Embedding External Knowledge: Incorporating external knowledge sources through APIs or databases allows the model to provide domain-specific information. This technique does not alter the model itself but enhances its output by integrating external data.
4. Fine-tuning: This technique is a deeper level of customization where the model is retrained on a smaller, task-specific dataset. This process adjusts the model's weights, allowing it to specialize in a particular domain or task. Fine-tuning requires access to computational resources and is often employed when prompt engineering or embedding external knowledge is insufficient.
5. Few-Shot Tuning: This technique involves providing the model with a small number of examples within the input prompt to guide its behavior or output. This approach does not alter the underlying model but instead optimizes the interaction by demonstrating the desired task or format.

In the context of this document, intelligent tools developed using generative AI in SE can include a wide range of applications, such as code generators, bug detection and fixing tools, test automation tools, code refactoring tools, documentation generators, and requirements engineering tools.

### **2.3 The ITLingo RSL Language**

The ITLingo RSL is a controlled natural language designed to improve the production of requirements specifications in a systematic, rigorous, and consistent manner [54]. RSL includes constructs logically arranged into views according to specific Requirement Engineering (RE) concerns, which are defined as linguistic patterns and represented textually by multiple linguistic styles. RSL provides several constructs logically classified

according to two dimensions: abstraction level and specific concerns they address. The abstraction levels are defined by business, application, software, and hardware. The following represent the concerns: active structure (subjects), behavior (actions), passive structure (objects), requirements, tests, other concerns, and relations & sets [54].

RSL constructs are logically classified according to two main dimensions: abstraction levels and specific concerns.

#### **Abstraction Levels:**

1. **Business:** This level focuses on high-level organizational requirements and goals. The constructs used here are generally more abstract and broad, providing a general overview rather than detailed specifications.
2. **Application:** The application level deals with the requirements specific to software applications. The constructs are more detailed than those at the business level and focus on the functionality and behavior of the application.
3. **Software:** This level specifies requirements for the software components and their interactions. The constructs at this level are highly detailed and technical, describing how software components should be implemented and interact.
4. **Hardware:** The hardware level pertains to requirements related to hardware components and their interactions. Similar in detail to the software level but focuses on physical components and their specifications.

#### **Specific Concerns:**

1. **Active Structure (Subjects):** Relates to the elements that perform actions or have roles within the system, such as actors or stakeholders.
2. **Behavior (Actions):** Describes the actions and behaviors that the system and its components must exhibit, including use cases and state machines.
3. **Passive Structure (Objects):** Pertains to the elements that are acted upon or used within the system, such as data entities and data clusters.
4. **Requirements:** Captures different types of requirements, including functional requirements, quality requirements, goals, and constraints.

5. **Tests:** Describes how requirements will be tested and validated, including test cases and acceptance criteria.
6. **Other Concerns:** Additional aspects such as glossary terms, risks, vulnerabilities, and stereotypes.
7. **Relations & Sets:** Defines relationships and groupings among the various constructs, ensuring a comprehensive and interconnected specification.

The relationship between abstraction levels and specific concerns in RSL ensures that requirements are captured comprehensively and systematically. Each abstraction level addresses concerns relevant to its scope, ensuring that high-level goals are aligned with detailed technical specifications. This structured approach enhances clarity, consistency, and traceability throughout the requirements engineering process, facilitating effective communication and implementation [55].

To illustrate these concepts, we present an RSL specification example:

The Library Management System is an application designed to manage the records of books and borrowers in a library. The system aims to help librarians keep track of the library's inventory.

A simple ITLingo RSL specification for such a system could be as follows:

```
Package LibrarySystem
System LibrarySystem "Library Management System" : Application [
  description "A system that manages books and borrowers in a
  traditional library." ]
Stakeholder stk_Librarian "Librarian" : Person [
  description "Librarian responsible for managing book records and
  borrowers." ]
Actor a_Librarian "Librarian" : User [
  stakeholder stk_Librarian
  description "Manages book records and borrowers." ]
DataEntity e_Book "Book" : Master [
  attribute ISBN : String [constraints (PrimaryKey)]
  attribute Title : String [constraints (NotNull)]
  attribute Author : String [constraints (NotNull)]
  description "Book entity storing information about library books." ]
UseCase uc_AddBook "Add Book" : EntityCreate [
  primaryActor a_Librarian
  dataEntity e_Book
  actions a_AddBookRecord
  description "Allows the librarian to add a new book to the library
  system." ]
ActionType a_AddBookRecord "Add Book Record" [
  description "Action to add a new book to the library system." ]
```

Interpreting the constructs for the abstraction levels, the categorization is the following:

- **Business Level:** In the example, the stakeholder (*stk\_Librarian*) could be considered as a concept introduced at the business level, where the roles and responsibilities of the librarian are broadly defined.
- **Application Level:** In the example, the system (*LibrarySystem*), actor (*a\_Librarian*), and use case (*uc\_AddBook*) are specified at the application level. These elements describe how the librarian interacts with the Library Management System to perform tasks like adding books.

The RSL categorizes constructs into different specific concerns based on what aspect of the system they address:

- **Active Structure (Subjects):** In the example, the actor (*a\_Librarian*) and stakeholder (*stk\_Librarian*) are defined as part of the active structure. They represent the user who will interact with the system to manage the library.
- **Behavior (Actions):** The use case (*uc\_AddBook*) and action (*a\_AddBookRecord*) are examples of behavior specifications. They describe the functionality provided by the system (adding a new book) and the specific action involved in carrying out that functionality.
- **Passive Structure (Objects):** In the example, the data entity (*e\_Book*) is part of the passive structure, representing the information stored about the books in the library.
- **Requirements:** The use case (*uc\_AddBook*) can be considered a functional requirement, describing a specific function that the system provides.
- **Tests:** Although this example does not specify any testing constructs. It could be added an acceptance test, "Acceptance Test for Add Book," that would verify the correct behavior of the "Add Book" functionality. It could specify preconditions, such as the librarian being logged in, the steps needed to execute the test, and the expected outcome of a successful book addition.
- **Other Concerns:** Although the example given does not explore these concerns explicitly, it could include a quality requirement that would specify a non-

functional requirement ensuring that the system performs the "Add Book" operation quickly.

- **Relations & Sets:** The example establishes relationships, such as the association between the actor (*a\_Librarian*) and the use case (*uc\_AddBook*), and between the data entity (*e\_Book*) and the use case (*uc\_AddBook*).

## 2.4 Artificial Intelligence Supporting Software Requirements Specification

The convergence of AI with SRS represents a frontier in software development, promising to address long-standing challenges associated with the traditional SRS process. By using AI's capabilities, particularly in natural language processing and machine learning, the process of eliciting and specifying requirements can be significantly enhanced [33].

AI-powered tools have the potential to improve SRS by automating the extraction and analysis of requirements from complex datasets [61], including unstructured data such as stakeholder interviews and documentation. These tools can provide clearer visions of user needs, reveal implicit requirements, and offer real-time validation of specifications against existing system constraints. Moreover, AI can facilitate the dynamic adaptation of SRS documents, ensuring they remain relevant and accurate throughout the software development lifecycle.

However, this integration is not without its challenges. The integration of AI in SRS raises questions about the balance between automated and human-driven processes [37]. There is a risk that over-dependence on AI could lead to less important or relevant critical thinking, experience-based intuition, and contextual understanding provided by human experts and a potential loss of nuanced understanding that comes from deep engagement with the requirements elicitation process.

Despite these concerns, the synergy between AI and SRS holds considerable promise. It offers a pathway to more efficient, accurate, and adaptable requirements specifications, ultimately leading to software systems that better meet the needs of their users and stakeholders.

As this chapter unfolds, it explores the current state of AI in SRS, examining the technologies at play, their practical applications, and the challenges and opportunities they

present. The aim is to provide a comprehensive understanding that informs both theoretical frameworks and practical implications for the future of software requirements specification.

# 3

## Literature Review

### **3 Literature Review**

This chapter introduces, analyzes, and discusses related work concerning the project's theme. A Systematic Literature Review (SLR) represents a rigorous and structured approach for reviewing existing literature relevant to the topic.

This SLR aims to provide a comprehensive overview of the current state of research on the topic, establishing a connection between artificial intelligence and software requirements specifications.

Following Kitchenham's "Procedures for Performing Systematic Reviews," the SLR for this study will be executed in three distinct phases [29].

The planning phase serves as the foundation of the SLR, ensuring the review's necessity and its potential to provide an objective understanding of AI's influence on software requirements specification.

During the conducting phase, the review protocol established in the planning stage was implemented. This includes systematically searching for and selecting studies that examine the application of AI in requirements specification.

The reporting phase is where the SLR findings are compiled and presented. The results will highlight current research trends, gaps, and the AI's potential to enhance software requirements specifications.

The SLR will be thoroughly documented at each stage to ensure the process remains transparent and the conclusions are well supported.

#### **3.1 Planning**

This chapter outlines the SLR's planning phase, highlighting the review's necessity and defining specific research questions. It details the development of a review protocol that includes selection criteria and identifies data sources and search strategies. This ensures a focused, systematic approach to gathering relevant studies.

### **3.1.1 Research Question**

This research intends to establish a connection between artificial intelligence and software requirements specifications, particularly the ways that affect the quality and precision of requirements specifications in software projects. Therefore, the following research question summarizes the scope of this study:

RQ: How does Artificial Intelligence (AI) improve requirements specification in software projects?

This question is divided into a set of sub-questions, focusing on different aspects of key areas of AI integration in the software requirements specification, and seeks to provide a comprehensive exploration of the topic for the systematic literature review.

SQ1: Impact on Requirements Specification Quality: How does the use of AI in requirements specification compare to traditional methods?

SQ2: Implications: What are the key challenges and benefits observed in software development processes due to the use of AI in requirements specification?

SQ3: Cost-Benefit and Efficiency Analysis: What are the cost implications, efficiency gains, and resource allocation differences in software projects when AI technologies are used for requirements specification?

SQ4: Impact on Project Outcomes and Success Rates: How does the use of AI in requirements specification correlate with software project completions?

SQ5: Adaptability and Scope of Application: In what types of software projects and under what conditions is the use of AI in requirements specification most effective and adaptable?

### **3.1.2 Study Selection Criteria**

The following criteria are set for the selection of the primary studies.

For the inclusion, the following criteria are considered:

- I1. Studies that preferentially address the use of Artificial Intelligence (AI) in the specification of software requirements;
- I2. Conference publications, journals, or industry reports focused on AI and requirement specification;

- I3. Considered relevant documents that have been reviewed by peers;
- I4. Articles written in English.

Exclusion criteria:

- E1. Publications that do not focus on the intersection of AI and requirement specification;
- E2. Grey literature, i.e. articles, without peer review;
- E3. Articles outside the context of software requirement specification;
- E4. Technical reports or research theses.

### 3.1.3 Literature Search Strategy

For this phase, the primary goal of the literature search was to identify academic studies at the intersection of artificial intelligence (AI)—including large language models (LLMs), natural language processing (NLP), and generative AI—with software requirement specifications (SRS) and requirements engineering (RE), using EBSCO and Scopus databases.

**The search strings:**

For **Scopus**:

TITLE-ABS-KEY (("Software Requirement Specification") OR ("requirements engineering"))  
AND (("AI") OR ("Artificial Intelligence") OR ("LLM") OR ("Large Language Model")  
OR ("Natural Language Processing") OR ("Generative AI"))

For **EBSCOHost**:

AB-Abstract (("AI") OR ("artificial intelligence") OR ("LLM") OR ("Large Language Model") OR ("Natural Language Processing ") OR ("Generative AI"))  
AND (("Software Requirement Specification") OR ("requirements engineering"))

**Boolean Operators:** Used to refine the search. “AND” is used to ensure that only documents containing both sets of keywords (related to AI and SRS/RE) are retrieved. “OR”

was used to include various synonymous terms and concepts within each set, broadening the search to capture all relevant literature.

**Search Fields:** “TITLE-ABS-KEY”, in Scopus, targets the title, abstract, and keywords of documents, ensuring a comprehensive retrieval of documents relevant to the research questions. Conversely, “AB-Abstract”, in EBSCOHost, focuses on the abstracts of documents, capturing detailed summaries of studies pertinent to AI applications in SRS and RE.

Although there are minor differences, both search strings meticulously target the described core concepts. A comprehensive protocol, detailing inclusion and exclusion criteria, search platforms, and search strings, guided this systematic review. This structured approach ensured methodological rigor and transparency, with the data collection process carried out by a single reviewer.

### **3.2 Conducting**

In the conducting phase of the SLR, the review protocol established in the planning phase was systematically executed to curate a collection of studies from the EBSCOHost and Scopus databases. This involved applying predetermined inclusion and exclusion criteria to the studies initially identified by the search strings.

#### **Filtering Process and Results:**

In Scopus, from 349 initially identified articles, applying filters for peer-reviewed status, English language, and specific document types (excluding books, chapters, notes, and reviews) resulted in 210 studies. Further refinement based on relevance and keyword alignment narrowed the selection to 83 studies for detailed analysis.

In EBSCOHost, 1132 studies were found. Application of filters (peer-reviewed, English language, academic journals, and removal of duplicates) narrowed this to 309. Further title and abstract screening excluded 83 studies, leaving 226 for detailed review.

#### **Iterative Review Process:**

The results from the screening of the studies were as follows:

Title and abstract review led to the exclusion of 100 studies in EBSCOHost and 30 in Scopus, with the remaining studies subjected to an introduction and conclusion review.

This subsequent review further narrowed the field: 80 studies were excluded from EBSCOHost and 15 from Scopus, based on relevance to research questions.

Studies were initially categorized as "Included", "Maybe" or "Excluded". All "Maybe" classified studies were later excluded for not aligning closely with the research focus on AI and requirements specification, resulting in the exclusion of 37 EBSCOHost and 29 Scopus studies after full-text review.

**Outcomes:**

The rigorous review process culminated in the selection of 18 relevant studies from both EBSCOHost and Scopus for the next stage of analysis.

A quality assessment framework was employed to evaluate the rigor and reliability of the selected studies. This framework considered factors such as search questions and the criteria established in planning. Figure 3.1 shows the conducting method applied and described before.

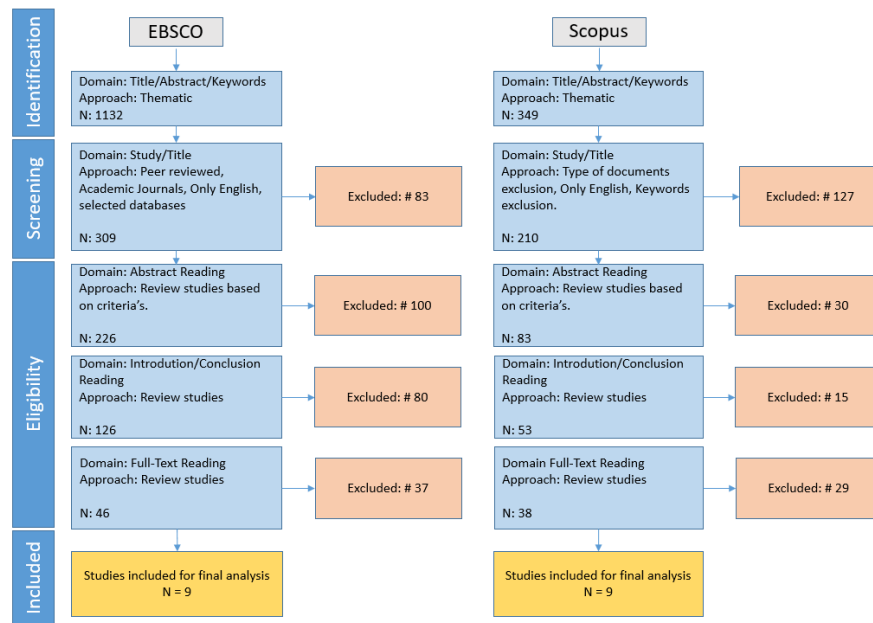
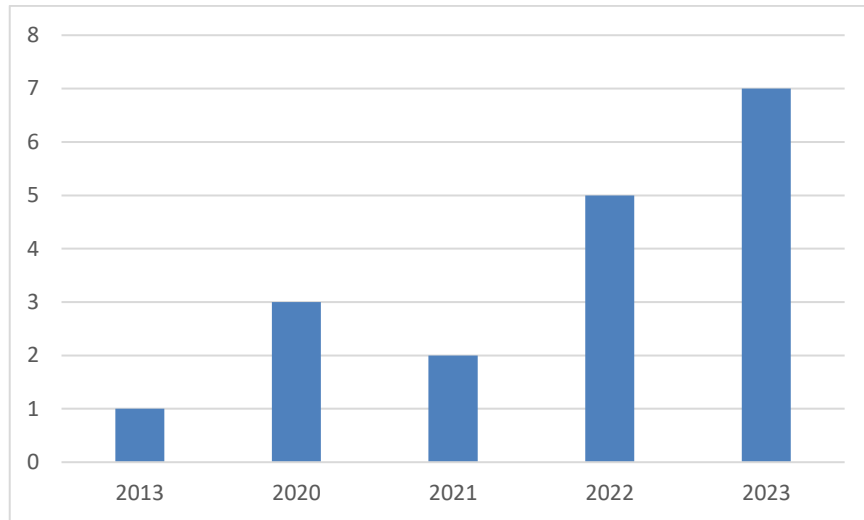


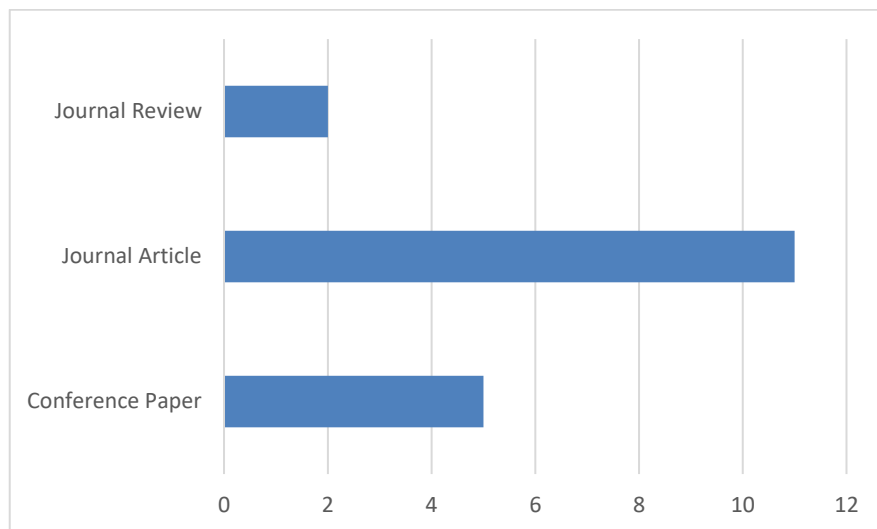
Figure 3.1 - Conducting stages of the review

The results of the included studies are organized in the year's timeline (Figure 3.2).



*Figure 3.2 – Number of studies by year*

The type of publications (figure 3.3) groups the results of the included studies.



*Figure 3.3 - Type of publication by studies*

In Appendix B, Table 10.12 shows the final set of the select studies by number, author, year, and type of publication.

### **3.3 Reporting**

This stage explores the documents by reviewing them in depth to establish conclusions and identify the connections between them. Main concepts related to the integration of AI

with SRS emerged from a detailed analysis of the selected articles, as summarized in Table 1. This table categorizes findings by relevant concepts, organized by the decreasing number of sources that mention them.

The integration of AI into the requirements specification process impacts multidimensionality, with each study offering unique contributions.

Table 3.1 - Main concepts related to studies

<i>Concepts</i>	<i>Studies</i>
<b>Foundational Concepts in AI and Machine Learning</b>	
<i>Artificial Intelligence or AI</i>	[S1, S2, S3, S4, S5, S7, S8, S9, S10, S11, S12, S13, S14, S15, S16, S17]
<i>Machine Learning</i>	[S2, S4, S5, S7, S9, S10, S11, S12, S14]
<i>Neural network</i>	[S12, S13, S15]
<i>Learning Algorithms</i>	[S4, S5, S10]
<i>Deep Learning</i>	[S5, S12]
<i>Machine learning techniques</i>	[S10]
<i>Support Vector Machines (SVM)</i>	[S9]
<b>Application of AI in Software Engineering</b>	
<i>Machine learning in requirements engineering</i>	[S4]
<i>Smart Requirement Engineering</i>	[S4]
<i>Automation of software requirement analysis</i>	[S11]
<i>Digitalization of requirement specification documents into processable models using AI</i>	[S16]
<b>Natural Language Processing</b>	
<i>Natural Language Processing</i>	[S1, S2, S5, S6, S12, S16, S18]
<i>Language Transformers</i>	[S5]
<b>Software Engineering Processes</b>	
<i>Software Requirements Specification</i>	[S1, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14, S15, S16]

<i>User Stories</i>	[S1, S12]
<i>Quality requirements</i>	[S11, S17]
<i>Software Development Life Cycle</i>	[S2]
<i>Requirements Validation</i>	[S4]
<i>Requirements Elicitation</i>	[S5]
<i>Usability Evaluation</i>	[S3]
<i>Validation Techniques</i>	[S4]
<i>Functional requirements</i>	[S1]
<i>Non-functional Requirements</i>	[S1]
<b>Technological Systems and Infrastructure</b>	
<i>Internet of Things</i>	[S13]
<i>Cloud Service</i>	[S11]
<b>Ethical and Social Aspects</b>	
<i>Decision making</i>	[S5, S13]
<i>Ethical Guidelines</i>	[S17]
<i>Human-computer interaction</i>	[S3]
<i>Knowledge acquisition</i>	[S7]
<b>User Feedback and Evaluation</b>	
<i>Consumer Reviews</i>	[S5]

Table 2 illustrates how AI and ML techniques automate and enhance the requirements elicitation, validation, and formalization processes. This table provides a comprehensive view of current methodologies, tools, and trends, laying the groundwork for future research directions and contributing to the development of more effective and efficient requirements engineering practices for AI-driven software development.

*Table 3.2 - Type of Requirements and Modeling Languages used by the study*

<i>Study</i>	<i>Type of Requirements</i>	<i>Used Techniques or Notations</i>
<i>S1</i>	Non-functional requirements	NLP, AI Techniques
<i>S2</i>	Software requirements, Requirement analysis	AI Techniques: NLP, Machine Learning

<i>S3</i>		Model-Based Engineering, Artificial Intelligence
<i>S4</i>	Usability requirements	AI-augmented Usability Evaluation Framework (AIUEF), OpenCog
<i>S5</i>	Functional requirements	Ensemble Machine Learning: Naïve Bayes, SVM, Decision Tree, Logistic Regression, SVC
<i>S6</i>	Software requirements, Requirements validation	Machine Learning Algorithms
<i>S7</i>	Quality requirements	Decision Trees (C4.5), Rule Lists (PART)
<i>S8</i>	Quality requirements	Multi-layer perceptron model (MLP), Convolutional neural network model (CNN)
<i>S9</i>	Non-functional (Privacy)	NLP tools (spaCy toolkit), Convolutional Neural Networks, Transfer Learning
<i>S10</i>	Software requirements, Requirements elicitation	Language Transformers, Zero-Shot Classification, Sentiment Analysis
<i>S11</i>	Software requirements, Formal requirements	NLP techniques, Dependency and POS-based Approach, Semantic Role Labeling
<i>S12</i>	The article focuses on the convergence of IoT and AI, which impacts a broad range of problems, including smart cities.	The article discusses the application of knowledge-enabled and data-driven AI in IoT.
<i>S13</i>	Functional and Non-functional	UML, Domain-Specific Modeling (DSM), Goal-Oriented Requirements Engineering (GORE)
<i>S14</i>	Functional and Non-functional	UML, SEMKIS-DSL
<i>S15</i>	Software requirements	Probabilistic Action-Model Learning (AMLCP algorithm)
<i>S16</i>	Software requirements	NLP Techniques, AI Engine (DizReq)
<i>S17</i>	Quality requirements, Ethical requirements	User Stories (for explainability requirements)
<i>S18</i>	Functional Requirements	Utilizes a combination of natural language processing (NLP) for initial data extraction and knowledge graphs for modeling relationships and entities

Incorporating analyses of Artificial Intelligence and Machine Learning integration into software engineering, particularly requirements engineering, within a Systematic Literature Review (SLR) is crucial. From reviewing Table 2, it was possible to infer certain trends. NLP is frequently used across various types of requirements, including non-functional and functional, privacy-specific, and formal requirements. General AI techniques are associated

with a wide range of requirements from software to usability and quality requirements. Machine Learning algorithms are mainly correlated with functional and quality requirements. Model-Based Engineering and UML are mentioned in the context of functional and non-functional requirements.

This approach aids in identifying trends, challenges, and research gaps in AI-driven software development methodologies. It enhances the SLR by providing insights into contemporary practices, applications, and the transformative influence of AI technologies on the software development lifecycle. This not only positions the research within the broader AI innovations in software engineering but also highlights the evolving landscape of requirements engineering in the context of AI and ML advancements. In Appendix B: Table 10.14 identifies the technologies and their correlation with the requirements engineering lifecycle. The following points describe the resume of the findings:

- **NLP and AI** are frequently used to automate the analysis of non-functional requirements and to structure software requirements from natural language user stories (S1, S2, S16).
- **Machine Learning and Deep Learning** techniques are applied to automate tasks such as validating software requirements, classifying functional and quality

requirements, and detecting privacy requirements within user stories (S4, S6, S7, S8, S9).

- **Model-Based Engineering** and **Development Operations (DevOps)** are integrated with AI to support the development of Cyber-Physical Systems, indicating an application of AI in the modeling stage (S3).
- **Usability Evaluation** and **Human-Computer Interaction (HCI)** are enhanced by AI frameworks, which are used to evaluate HCI requirements in SRS without needing end-user involvement (S5).
- **IoT and AI** are combined to improve capabilities in IoT architecture, which can impact a range of software requirement areas (S12).
- **UML and Domain-Specific Modeling Languages** are used to specify and model requirements in AI systems, suggesting their application in functional and non-functional requirements (S13).
- **Automated Planning and Machine Learning** are utilized for transforming software requirements into formal specifications, indicating a broad application across various types of requirements (S15).
- **Ethical Guidelines Analysis** and **Transparency** approaches are used to transform ethical considerations into practical requirements for AI systems, focusing on ethical and explainability aspects (S17).
- **BiLSTM, CRF, and Knowledge Graphs** are employed for sequential data processing, pattern recognition, and data structuring, indicating a method for improving the precision of software requirements (S18).

Reflecting on the findings incorporated in Tables 1, and 2 above and in Appendix B: Table 10.14, it is evident that new technologies and methodologies, particularly those based on AI and ML, are set to play or are already playing significant roles in software engineering. This is especially true in the domain of SRS, where AI components, including machine learning, deep learning, and various algorithmic approaches, are being integrated to define, analyze, and improve the quality and usability of requirements.

In the software development lifecycle, requirements engineering serves as the foundational phase where stakeholder's needs are systematically identified and analyzed, culminating in the creation of a detailed SRS document [7]. After validation, this document then informs the software design phase, guiding the architectural and detailed design decisions necessary to fulfill the specified requirements. The software requirements specification can be described as a complete description of the behavior of the system to be developed. It includes a set of models and a collection of use cases describing all the interactions with the system. They are divided into functional requirements, inferring functionalities or system services, and non-functional requirements, relating to system properties and constraints.

In this matter, the studies clearly show that technologies and methodologies—ML, NPL, or Deep Learning based—are being integrated into various systems and stages of software engineering. Specifically in the software requirements specification, in the definition of functional and non-functional requirements, analyses and quality of requirements, and in the usability of requirements.

The artificial intelligence components used from system to system differ. It was possible to establish that most of the systems use machine-learning components, some relying on decision trees, support vector machines (SVM), or linear regression. Others use deep learning, which is a subset of machine learning that heavily relies on neural networks [8]. Even though some cases of deep learning use Transformer Models (RNNs or CNNs), which in principle seem like a type of neural network, their architecture and capabilities differ significantly from more traditional models [8].

For further understanding, figure 4 represents AI structure and relationships between Artificial Intelligence, Machine Learning, and Deep Learning [8].

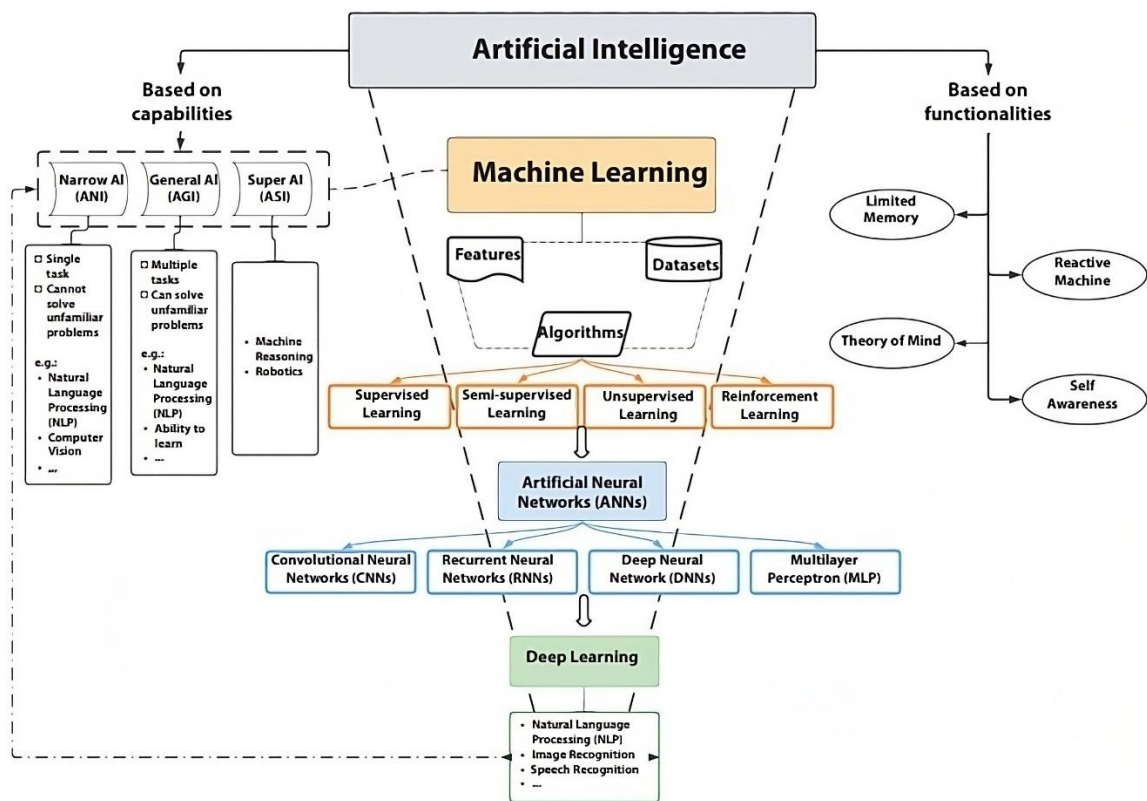


Figure 3.4 – Diagram representing the relationships between Artificial Intelligence, Machine Learning, and Deep Learning (extracted from Bellini et al., 2022, p. 3) [8]

The current state of the art in AI, as observed from the selected studies, shows a strong engagement with machine learning (ML), deep learning (DL), and natural language processing (NLP) technologies. Researchers use these tools to enhance systems' capabilities to understand, process, and generate human-like responses to complex data sets. AI's applications range from predictive analytics and automated decision-making to natural language understanding, which are crucial for processing large volumes of unstructured data, identifying patterns, and making informed decisions based on historical data insights.

In the domain of SRS, the present research reflects a concerted effort to capture, analyze, and manage software requirements with greater precision and adaptability. The focus is on methodologies that facilitate clear, comprehensive, and traceable requirements to guide the development process effectively. Directions for future research in this area highlight the need for tools and approaches that can accommodate the dynamic nature of software projects,

ensuring that requirements are kept up to date and remain aligned with stakeholder needs and system constraints.

The following section addresses the key questions established during the planning phase, offering a comprehensive understanding and analysis based on the collected research and relevant studies.

### **Sub-question 1: "Impact on Requirements Specification Quality"**

The selected studies indicate that AI significantly enhances the quality of requirements specification. For instance, AI-driven frameworks and tools, such as SEMKIS-DSL (S15) or the BiLSTM-CRF Model (S18), are introducing more structured, precise, and automated approaches to requirements specification, significantly enhancing the quality and precision of the output. The use of AI for non-functional requirements in agile projects clarifies and completes specifications, improving project success and quality (S1). Similarly, AI-based techniques in requirement analysis identify and resolve analysis issues, offering wide applicability across different engineering projects (S2). Moreover, an AI-augmented usability evaluation framework specifically enhances the accuracy of HCI requirements in SRS (S3). Techniques like NLP (S1, S2, S16, and S18) and machine learning (S1, S2, S3, S4, S6, S7, S8, S9, S10, S11, S14, S15, and S16) are used to accurately identify and classify requirements, reducing manual effort and the potential for human error. These findings collectively demonstrate AI's potential to refine and advance the specification quality beyond traditional methods, ensuring more precise, clear, and comprehensive requirements across diverse software projects.

### **Sub-question 2: "Implications: Challenges and Benefits"**

The synthesis reveals that AI integration in requirements specification brings both key challenges and significant benefits to software development processes. The use of ensemble methods like Naïve Bayes, SVM, and Decision Trees has led to an accuracy rate of 99.45%

in classifying functional requirements, showcasing the potential for AI to enhance precision in the requirements specification. AI and NLP have automated the analysis of non-functional requirements from user stories (S1), which reduces manual effort and speeds up the requirements inspection process. Machine learning algorithms have been applied to automate the validation of software requirements, as seen in the study where prototyping and validation techniques were employed (S4). This approach can improve the accuracy of requirements validation and reduce the reliance on manual validation methods. The reliance on AI for requirements specification raises challenges about the balance between automated and human expertise (S12). The studies highlight the challenges in the complexity of integrating AI into existing development operations and model-based engineering (S3). This complexity can stem from the need to adapt AI tools to fit into established workflows and the requirement for stakeholders to understand and trust AI-driven decisions. The transformation of ethical guidelines into practical requirements (S17) points to the challenge of ensuring AI systems are transparent and adhere to ethical standards.

### **In sub-question 3: "Cost-Benefit and Efficiency Analysis"**

The reviewed studies reveal that AI integration in requirements specification significantly influences cost implications, efficiency gains, and resource allocation. For instance, automation in the validation process and requirements elicitation from consumer data are expected to offer substantial cost savings and efficiency improvements (S4 and S5). Advanced AI techniques such as deep learning (S1, S8, and S9) and transfer learning (S9) also contribute to a more efficient processing and analysis of requirements, potentially reducing costs associated with privacy breaches and ensuring compliance. AI's ability to streamline the requirements engineering process, reduce manual effort, and enhance accuracy leads to a favorable cost-benefit scenario, optimizing resource utilization across diverse project contexts.

#### **Sub-question 4: "Impact on Project Outcomes and Success Rates"**

The reviewed studies clarify that integrating AI in requirements specification significantly correlates with enhanced project outcomes and success rates. The studies show that by refining requirements through AI technologies—such as automating requirements validation or generating requirements from data—projects can achieve higher quality and completion rates. For example, automation in the requirements validation process (S4) and the use of machine learning techniques for quality assessment (S2, S4, S5, S7, S9, S10, S11, S12, and S14) are highlighted as contributing to improved project success. These findings suggest that AI's role in accurately defining and validating requirements directly influences the likelihood of project success, underscoring the critical impact of AI-enhanced specifications on the overall software development lifecycle, ensuring a more informed decision-making process, and leading to higher success rates and user satisfaction.

#### **Sub-question 5: "Adaptability and Scope of Application"**

The analysis across studies indicates that AI's integration into software requirements specification exhibits extensive adaptability and applicability across various software projects. The studies showcase AI's versatility in different contexts, from enhancing non-functional requirements in agile environments (S1) to facilitating requirements elicitation through automated data analysis (S10). Moreover, AI's application spans diverse domains and different layers of technology, including cyber-physical human systems (S4) and IoT architecture (S12), demonstrating its broad utility in addressing specific project needs and environments. This adaptability underscores AI's potential to revolutionize requirements engineering practices across the spectrum of software development.

These assessments reflect the transformative potential of AI in enhancing the requirements specification process in software projects, promising not only improved precision and quality but also operational efficiency, cost-effectiveness, enhanced project outcomes, and wide applicability across various domains.

The objective of this investigation is to capture the essence of the findings from the documents reviewed, correlating them with the theme of this Systematic Literature Review (SLR) on "How does the integration of Artificial Intelligence (AI) affect requirements specification in software projects?" The findings indicate that AI technologies, particularly natural language processing, can significantly enhance the accuracy of software requirements. However, challenges such as the need for large datasets and domain-specific training remain prevalent.

The analysis aimed to highlight the key elements from the documents in relation to the impact of AI integration on various aspects of requirements specification in software projects, ensuring a comprehensive overview aligned with the theme of your SLR.

### **3.4 Discussion**

The integration of AI into software requirements specification projects emerges as a transformative force, reinventing traditional methodologies and catalyzing a shift towards more dynamic, precise, and user-centric development processes. The findings from answered questions underscore a trend towards automation, enhanced accuracy, and operational efficiency, revealing AI's key role in advancing the quality and precision of requirements specification. This trend is not merely a reflection of technological advancement but a paradigm shift that underscores the synergy between AI and human expertise, fostering a more collaborative, informed, and adaptive approach to software development.

This analysis highlights the significance and relevance of using AI as a tool, emphasizing the need to develop new methods for integrating AI with software requirement specifications. In this context, artifacts such as chatbots can facilitate this connection by acting as mediums that collect requirements and convert them into a universal requirements specification language. This process can overcome eventual ambiguities, errors, and mistakes derived from the initial specification.

Advances in the field of AI have grown exponentially, we can observe an increasing number of integrations across diverse fields. These include virtual assistants like Siri and

Alexa, recommendation systems in e-commerce, fraud detection in financial institutions, autonomous vehicles, NLP applications for chatbots and customer service, image and facial recognition systems in security contexts, as well as medical diagnosis and healthcare systems. The applicability is extensive, positioning AI as a powerful tool for addressing a wide range of practical challenges.

For the purpose of this study, the SRS process, while capable of being aligned with established rules and frameworks to improve its outcomes, remains heavily reliant on human input, which can introduce potential flaws. Empirical research, particularly through case-based studies, could offer valuable insights into the practical challenges and benefits of integrating AI into the SRS process, thereby deepening our understanding of its real-world applications.

In a recent study, Wagner Stefan et al. (2016) tried to identify how industrial environments conduct requirements engineering in an agile setting and what problems they face [60]. The mapping of this information, on how requirements are elicited and documented, how respondents deal with changing requirements, why and how RE can be improved, and on common RE problems, made available important data, which, most certainly, being systematized, could be used in AI as a predictive model.

As more information is gathered and documented about the processes, the systematization of AI as a predictive model or acting upon the different stages of requirements engineering will enhance the role of AI in software requirements specification, in upscaling on requirements engineering, and on a broader scale, software engineering.

In this context, the following themes should see improvement in the near future.

**Advanced Predictive Analytics** can power AI to predict future trends and user needs, enabling developers to proactively tailor software solutions. This approach could lead to the development of more anticipatory and user-responsive software systems. [39].

**Augmented Decision-Making** weights AI to offer deep insights and recommendations, thereby enhancing the decision-making process in software development. This could lead to more strategic and data-driven choices, significantly reducing the time and resources spent on revisions and error rectifications [16].

**Customization at Scale** refers to AI's capability to handle and analyze vast datasets, which can be harnessed to offer highly personalized user experiences. Software projects can benefit from this by delivering more configured solutions tailored to individual user preferences and requirements [2].

**Enhanced Collaboration** refers to AI acting as a bridge between various stakeholders in a software project, ensuring that requirements are communicated, understood, and implemented accurately. This can lead to more cohesive and synergistic development efforts [64].

**Ethical and Responsible AI** is becoming increasingly integrated into requirements specification, leading to a growing emphasis on developing ethical AI frameworks that respect user privacy, ensure data security, and promote transparency [36].

**Learning and Adapting Systems** refers to future solutions that are not only built to specifications, but can also learn and adapt after deployment based on user interactions and feedback, allowing software to evolve and improve over time [47].

In conclusion, this SLR highlights the promising role of AI in enhancing software requirements specification, offering significant improvements in accuracy and efficiency. However, it also underscores the need for continued research to address current challenges and evolve these technologies for broader applications. The AI integration with requirements specification is set to redefine the horizons of software development. By harnessing the power of AI, the field can look forward to not only mitigating current challenges but also unlocking new opportunities for innovation, customization, and excellence. The ongoing evolution of AI capabilities, coupled with a deepening understanding of its potential, is poised to usher in a new era of software development characterized by heightened efficiency, adaptability, and user alignment. This study contributes to the growing body of knowledge in AI applications within software engineering, providing a foundation for future research in this important area.

# 4

## **The GenAI4RSL chatbot**

## 4 The GenAI4RSL chatbot

Addressing the challenges and opportunities presented by the integration of AI in SRS, our work developed the GenAI4RSL, a chatbot-based framework designed to enhance and streamline the traditional SRS process. Enhancing the latest AI version of GPT-4 capabilities [31], GenAI4RSL aims to improve how software requirements are captured, interpreted, and formalized, consistent with recent research identifying GenAI's transformative potential in requirements engineering [14].

The proposed solution centers on automating the translation of NL inputs into formalized SRS documents, addressing common issues such as ambiguity, inconsistency, and inefficiency that often plague traditional methods. The proposed chatbot employs state-of-the-art GPT algorithms to parse complex linguistic constructs. It generates clear, unambiguous requirements with a high degree of accuracy. This reduces the time and effort required to produce SRS documents. Also, decreasing the risk of misinterpretation and errors in subsequent stages of software development, aligning with findings on the impact of AI on improving accuracy and automation in requirements specification [32].

GenAI4RSL is designed to ensure compatibility and ease of adoption within diverse software development environments. This integration potentially shifts the SRS's landscape by enabling affordable, accessible AI-driven solutions like GenAI4RSL to bridge the gap between informal natural language requirements and formalized counterparts. By doing so, the solution offers a more efficient, accurate, and scalable approach to requirements specification, ultimately leading to better-defined software projects and more successful outcomes.

Through this proposal, we intend to demonstrate the GenAI4RSL potential to improve the efficiency and effectiveness of SRS processes in the ever-evolving field of software development.

#### **4.1 Artificial Intelligence chatbot GenAI4RSL as a Solution**

OpenAI's text generation models, known as Generative Pre-trained Transformers (GPT), in their current version GPT-4, have demonstrated potential in addressing challenges in SRS. This aligns with recent studies that emphasize the role of AI in improving requirements engineering (RE) processes, highlighting the potential of generative AI models in facilitating more accurate and efficient requirement elicitation and analysis [14].

Incorporating AI-based systems into RE processes introduces challenges, such as managing inconsistencies and ensuring the output quality, which the GenAI4RSL solution seeks to address. This is supported by findings from recent research, which indicated that the automation of RE tasks through AI can significantly improve the efficiency of the specification process while addressing common issues like ambiguity and inconsistency [59]. By resourcing to structured patterns inherent in the RSL, GenAI4RSL tries to mitigate issues such as hallucinations that can occur with AI-generated outputs, ensuring a more robust and consistent approach to requirements specification.

Focusing on the objective of this work, the purpose is to customize the GenAI4RSL chatbot with specific functionalities aimed at optimizing the requirements engineering process, especially in the requirements specification.

Therefore, the primary goal in constructing this chatbot is to critically examine the impact of artificial intelligence on software requirements specification. To this end, the GenAI4RSL is designed to fulfill the following key functions:

1. Translate software requirements specified in natural language into a rigorous RSL specification.
2. Convert RSL specifications back to structured natural language.

The GenAI4RSL framework aims to enhance the requirements engineering process by utilizing RSL to generate software requirements systematically. This approach reduces ambiguities and errors that could negatively impact later stages of software development [54]. By structuring specifications across various abstraction levels, GenAI4RSL effectively

translates functional and non-functional requirements [55]. Additionally, employing RSL helps mitigate “hallucinations”<sup>1</sup>—incorrect or irrelevant outputs—commonly produced by AI models like GPT-4.

The language is designed to be process- and tool-independent, making it adaptable to various methodologies and software tools. It can be used in different styles depending on the project's needs, from high-level overviews to detailed specifications, supporting both textual and graphical representations.

In this context, the RSL language is perceived as an essential tool in helping the chatbot transform natural language requirements into formalized RSL specifications by:

### 1. **Natural Language Understanding:**

- **Controlled Vocabulary:** RSL employs a controlled natural language, reducing ambiguity and improving the clarity of requirements. The GenAI4RSL can use this controlled language to parse and interpret user inputs more effectively.
- **Linguistic Patterns:** The use of predefined linguistic patterns, allows the GenAI4RSL to map natural language statements to specific RSL constructs, ensuring that the requirements are captured accurately.

### 2. **Rigorous Specification:**

- **Systematic Structuring:** By guiding users through the structured approach of RSL, GenAI4RSL can ensure that requirements are systematically specified, covering all necessary aspects such as actors, actions, and data entities.
- **Consistency and Validation:** RSL’s formal structure allows the chatbot to automatically validate the consistency of requirements, identify missing elements, and suggest corrections, leading to higher quality specifications.

### 3. **Traceability and Flexibility:**

---

<sup>1</sup> Although there is not one universal definition for this phenomenon, we can take the following definition in the Academic context: “AI hallucination” refers to the phenomenon where ChatGPT may generate output that is grammatically correct and coherent but may not be appropriate for the intended audience or purpose.” (Piazza et al., 2023)

- **Traceability:** RSL’s ability to establish relationships between different constructs (e.g., use cases, data entities, state transitions) allows the chatbot to maintain traceability throughout the requirements specification process, ensuring that all aspects are interconnected and aligned with the project’s goals.
- **Flexible Specification Levels:** Depending on the user's input, the chatbot can generate RSL specifications at varying levels of detail, from broad overviews to detailed, step-by-step processes.

Figure 4.5 illustrates the workflow of the GenAI4RSL-based training:

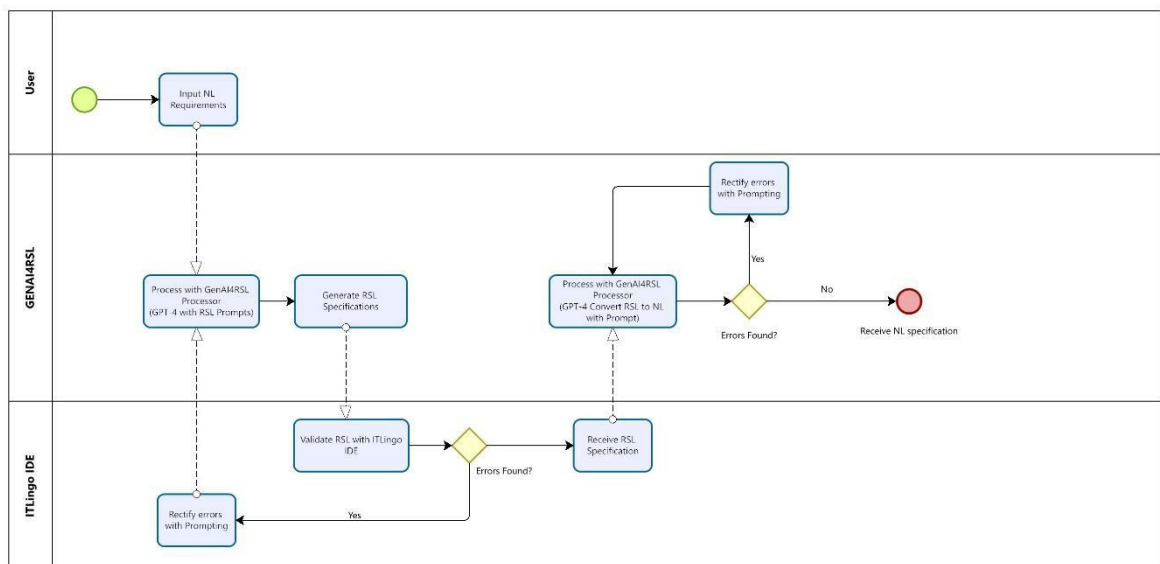


Figure 4.5 - GenAI4RSL-based workflow

## 4.2 Data Collection & Pre-processing

This stage outlines the steps necessary to customize GenAI4RSL. By providing instructions and examples from a dataset, we fulfill the first phase of our objective. The importance of tailoring a chatbot to enhance performance has been emphasized in various studies. For instance, Wangsa et al. (2024) demonstrated that the effectiveness of chatbots

like ChatGPT, Gemini, Llama, Ernie, and Grok is significantly influenced by their ability to be fine-tuned using human instructions [63].

The primary goal of GenAI4RSL's instructions is to define the model's capabilities and guide users on how to interact with it effectively. These instructions are essential in guiding users to utilize the GPT's functionalities to their fullest, in this case for translating requirements specified in NL to RSL language and back into a set of structured NL requirements [46].

Customization of GenAI4RSL involves a set of tailored instructions that organize the system's operational framework into four distinct categories: Role and Goal, Constraints, Guidelines, and Clarification & Personalization. Each category serves a specific purpose, ensuring that GenAI4RSL functions efficiently within its intended scope and delivers outputs that are precise and practical for software development processes.

This categorization is not arbitrary; rather, it emerges from a synthesis of existing research and conversational AI methodologies. Studies on persona-driven design highlight the importance of defining clear roles for AI systems to ensure alignment with user expectations [35]. Advances in personalized dialogue systems demonstrate how clarification and user-specific adaptation can enhance interaction quality [13].

The framework presented in this research synthesizes these principles into a cohesive structure tailored to the GPT-based development systems for specialized domains such as software development. By categorizing chatbot instructions as a specification of functional requirements, resulting in these four key areas, we aim to develop a more robust and efficient conversational AI system that effectively meets the specialized needs of software development processes.

The set of instructions is explained as follows:

### **Role and Goal**

The GenAI4RSL functions as a requirements engineer tasked with translating NL requirements into RSL and back. This process aims to facilitate the production of a comprehensive final report detailing the intended requirements.

## **Constraints**

**Explanation:** Constraints establish the limitations and boundaries from which the chatbot must operate.

**Focus:** The GenAI4RSL should concentrate solely on interpreting and translating NL requirements into RSL specifications and back.

## **Guidelines**

**Explanation:** Guidelines provide detailed instructions and best practices on how the chatbot should perform its tasks. They offer direction on preferred methods, formatting standards, and stylistic conventions.

### **Focus:**

- **Tone and Accuracy:** Maintain a formal and technical tone, ensuring accuracy and clarity in the translation.
- **Clarification:** Seek clarification selectively when requirements are ambiguous or incomplete.
- **Improvement Suggestions:** Provide suggestions for improving the clarity and detail of the requirements when necessary.

## **Personalization**

**Explanation:** This category encompasses instructions that allow the chatbot to adapt to specific user needs and handle specialized tasks. By including personalized instructions for converting natural language to RSL and RSL to natural language, we enhance the chatbot's ability to perform specialized translations effectively, ensuring that outputs are relevant, precise, and useful to users within the software development process.

### **Focus:**

**RSL Constructs:** Utilize RSL constructs appropriately across different abstraction levels (business, application, software, hardware) and specific concerns (active structure, behavior, passive structure, requirements, tests, other concerns, relations & sets).

**RSL to NL Conversion:** Accurately translate RSL constructs into clear and structured natural language requirements. Ensure that the translations are understandable to stakeholders with varying levels of technical expertise, maintaining consistency in terminology and adhering to domain-specific language conventions across different abstraction levels (business, application, software, hardware) and specific concerns (active structure, behavior, passive structure, requirements, tests, other concerns, relations & sets).

### Summary of Key Constructs

To aid in the translation process, there were provided summaries and syntax examples of key RSL constructs, such as DataEntities, Actors, UseCases, StateMachines, Events, and Views. As shown in Table 4.3, the key constructs for the RSL framework are categorized based on specific concerns and abstraction levels.

Table 4.3 – RSL’s Key Constructs

<i>Construct</i>	<i>Summary</i>	<i>Syntax</i>
<i>DataEntity</i>	Represents data concepts derived from domain analysis.	<code>DataEntity: 'DataEntity' name=ID (nameAlias=STRING) ? ':' type=DataEntityType [...]</code>
<i>Actor</i>	Represents participants interacting with the system.	<code>Actor: 'Actor' name=ID (nameAlias=STRING) ? ':' type=ActorType [...]</code>
<i>UseCase</i>	Sequences of actions performed by actors.	<code>UseCase: 'UseCase' name=ID (nameAlias=STRING) ? ':' type=UseCaseType [...]</code>
<i>StateMachine</i>	Define the behavior of DataEntities or Clusters indicating states and transitions.	<code>StateMachine: 'StateMachine' name=ID (nameAlias=STRING) ? ':' type=StateMachineType '[' 'dataEntity' entity=[DataEntityGeneric   QualifiedName] states= States '[' States: {State} states += State* State: 'state' name=ID (nameAlias=STRING) ? (isInitial?= 'isInitial') ? (isFinal?= 'isFinal') ? ':' ...transitions... Transition: (ucAction= RefUCAction ('nextState' nextState= [State]) ? RefUCAction: 'useCase' useCase=[UseCase   QualifiedName] 'action' action=[UCAction   QualifiedName]</code>

<i>Events</i>	Represent specific occurrences that may trigger actions or state changes	Event <event_id> "<event_name>" : <event_type> [isCatch]
<i>Views</i>	Provide a visual representation of the system's structure, relationships, and functionality	View <view_id> "<view_name>": <view_type> [ <elements> description "<description>"]

Table 4.3 illustrates the core constructs of the RSL. Each construct plays a crucial role in ensuring systematic and consistent SRS.

The set of instructions for the GenAI4RSL is designed to enhance the process of specifying requirements by grouping them at various abstraction levels and specific concerns. This approach provides clarity and reduces ambiguity in the requirements specification process, allowing the requirements to be rewritten in natural NL as a structured document, free from ambiguities, uncertainties, and errors.

The next stage consisted of training and collecting data to form the base knowledge, crucial for Gen4AIRSL functionality in translating software requirements in natural language into rigorous specifications using the RSL. This data collection is paramount for building the GenAI4RSL knowledge set, enhancing its capabilities in understanding and processing user inputs accurately and efficiently. For the chatbot to interpret and convert natural language into RSL, a comprehensive dataset was assembled, which included a variety of requirements previously specified in RSL language and originated from other projects. These datasets were equipped with examples of RSL constructs with respective syntax and semantics, enabling GenAI4RSL to learn the structure and rules necessary to generate valid RSL requirements from the inputted natural language specifications.

Building a knowledge base was crucial for GenAI4RSL to interpret and convert natural language accurately into RSL. A comprehensive dataset was collected, including RSL requirements specified in different system projects. Below are listed the datasets included in the knowledge base.

Table 4.4 - Training Dataset for the GenAI4RSL

<i>Dataset</i>	<i>Description</i>	<i>Source</i>	<i>Lines</i>	<i>Elements</i>	<i>Types of elements</i>	<i>Reference</i>
<i>User Management System</i>	The user management system must allow users to create, edit, and delete their profiles, and should send a confirmation email after each action.	GenAI4RSL, ITLingo IDE	99	7	Stakeholders Actors DataEntities DataEntityCluster Actions UseCases StateMachine Non-Functional Requirements	Appendix D1
<i>Billing System</i>	BillingSystem is a system that allows users to manage customers, products, and invoices.	[54]	340	8	Event View DataEntities DataEntityCluster Data view Actions Stakeholders Actors UseCases	Appendix D2
<i>Library Management System</i>	LibraryManagement is a system designed to manage the records of books and borrowers in a library.	GenAI4RSL, ITLingo IDE	181	9	DataEntities DataEntityCluster Stakeholders Actors Actions UseCases StateMachine EventView Non-Functional Requirements	Appendix D3
<i>Appointment Scheduler</i>	Appointment Scheduler System for scheduling	GenAI4RSL, SL,	122	5	DataEntities Actions Stakeholders	Appendix D4

<i>dulerSystem</i>	and managing appointments for a service provider	ITLingo IDE			Actors UseCases	
<i>ManagementAccountingSystem [RequirementsReviews]</i>	ManagementAccounting is a system that allows the import, visualization, and editing of revenue and transaction items through the management accounting structure.	Project: Management ent Accounti ng	390	8	EventView DataEntities DataEntityCluster Stakeholders Actors Actions UseCases StateMachine	Appendix D5

This dataset includes translations in RSL and NL specifications generated from the GenAI4RSL and refined in ITLingo IDE. This dataset enables the model to learn the necessary structures and rules in RSL and NL specifications.

The documents generated from the dataset in the table specified in RSL language were built in the initial training by constructing different systems from theory and real-world software projects.

### 4.3 Initial Training

This process was focused on enabling the GenAI4RSL to understand NL inputs and start generating RSL specifications.

For this task, it was expected that the GenAI4RSL would be able to understand basic software requirements and generate initial SRS documents in RSL, laying the groundwork for further refinement through iterative prompt-engineering.

During the initial training phase, the focus was on tailoring the GPT-4 model to accurately interpret and generate RSL specifications. Although GPT-4 is a powerful language model, it requires careful prompt engineering to handle domain-specific languages like RSL. There

were designed prompts that included explicit instructions, examples of correct RSL syntax, and common pitfalls to avoid.

To fine-tune the model's performance, it was:

- **Developed Structured Prompts:** Crafted prompts that delineated the task, provided context, and specified the desired output format. The set of these prompts is presented in Appendix C.
- **Iterated on Feedback:** Collected the model's outputs, identified errors, and adjusted the prompts accordingly to correct misunderstandings. This phase is described in more detail in the next paragraphs.

While traditional fine-tuning (modifying the model's weights with additional training data) was not performed, GPT-4's capability for in-context learning was used through prompt engineering.

The strategy for customizing and building the knowledge base for the GenAI4RSL was to construct the dataset described in Table 4.4 using prompt engineering.

The first step consisted of translating NL requirements into RSL specifications. The prompts used for each of the systems are described in Appendix C for each of the systems.

The initial tests in translating NL requirements into RSL specifications for the User Management System definition resulted in errors, mainly in syntax constructs and the inclusion of different symbols compared to the syntax given in the instructions. The errors were identified by running each RSL produced by the GenAI4RSL in the ITLingo IDE. After the manual correction in the IDE, the code was included in GenAI4RSL as knowledge and prompt correction.

In addition to the instructions, the GenAI4RSL customization focused mainly on two iterative processes:

1. Rectifying errors or inaccuracies that arise during NL requirements translation into RSL specifications.

**Process implementation:**

- **Error Analysis:** Collect instances where the model's output is incorrect or suboptimal.
- **Data Correction:** Include corrected examples in the fine-tuning dataset.

- **Data knowledge:** Include the corrected specifications to allow the chatbot learning process.
2. Prompting the request for RSL specifications to be translated to NL and providing feedback on the generated output.

This process establishes a feedback loop between the user and the chatbot, which is essential for refining the chatbot's performance and ensuring it meets user needs.

### **Error Analysis:**

The errors were categorized into the following types:

1. **Syntax Errors:** Misuse of RSL syntax.
2. **Semantic Errors:** Incorrect interpretation of the requirements leading to inaccurate RSL representations.
3. **Omissions:** Failure to include all necessary components of a requirement, such as missing data entities or actions.
4. **Hallucinations:** Introduction of irrelevant or incorrect information not present in the input.

From the next NL requirement specification:

*"The system must allow users to create, edit, and delete their profiles and should send a confirmation email after each action."*

The translation to the RSL specification generated the following types of errors:

### **Error Analysis**

#### **1.1. Syntax Error**

Description: The attribute `ID` is missing a colon between the attribute name and the data type:

```
...
DataEntity e_UserProfile "User Profile": Master [
attribute ID Integer [constraints (PrimaryKey)]
attribute email : Email [constraints (NotNull Unique)]
attribute name : String [constraints (NotNull)]
attribute password : String [constraints (NotNull)]
description "DataEntity to represent user profiles." ]
```

...

### Semantic Error

Description: In UseCase uc\_ManageUserProfile, an incorrect precondition is added.

```
...
UseCase uc_ManageUserProfile "Manage User Profile": EntitiesManage [
  primaryActor a_User
  dataEntity e_UserProfile
  actions a_CreateProfile, a_EditProfile, a_DeleteProfile,
  a_SendConfirmationEmail
  precondition "User must be an administrator."
  description "Allows users to manage profiles only if they are
  administrators, and sends a confirmation email after each action." ]
```

This precondition contradicts the original requirement, which states that **end users** should be able to manage their profiles. The original use case is intended for all users, not just administrators. In addition, the statement precondition should come before actions.

### Omission Error

Description: The ActionType a\_SendConfirmationEmail and its associated functionality are missing.

```
...
ActionType a_CreateProfile "Create Profile" [
  description "Allows creation of a new user profile." ]

ActionType a_EditProfile "Edit Profile" [
  description "Allows editing an existing user profile." ]

ActionType a_DeleteProfile "Delete Profile" [
  description "Allows deletion of a user profile." ]

UseCase uc_ManageUserProfile "Manage User Profile": EntitiesManage [
  primaryActor a_User
  dataEntity e_UserProfile
  actions a_CreateProfile, a_EditProfile, a_DeleteProfile
  description "Allows users to manage profiles only if they are
  administrators, and sends a confirmation email after each action." ]
```

### Hallucination Error

Description: The translation generates the following set of elements:

- New Actor: Actor a\_SupportAgent "Support Agent"

- **New ActionType:** ActionType a\_PasswordRecovery "Password Recovery"
- **New UseCase:** UseCase uc\_PasswordRecovery "Password Recovery Process"

```

...
Actor a_SupportAgent "Support Agent": Person [
description "Support agent assisting users." ]

ActionType a_PasswordRecovery "Password Recovery" [
description "Allows users to recover forgotten passwords." ]

UseCase uc_PasswordRecovery "Password Recovery Process": EntitiesManage[
primaryActor a_User
secondaryActor a_SupportAgent
dataEntity e_UserProfile
actions a_PasswordRecovery
description "Enables users to recover forgotten passwords with support
agent assistance." ]

```

## 1.2. Data Correction

For each error detected and manually corrected in the ITLingo IDE, the following prompt was used to teach the model to:

- Recognize different types of errors.
- Apply corrections to produce accurate and error-free RSL code.

Prompt for “Syntax Error”:

```

The following RSL specification contains an error:

System UserManagement "User Profile Management": Application [
description "System to manage user profiles including creation,
editing, deletion, and email notifications."
]

Actor a_User "User" : User [
description "End user managing their profile."
]

DataEntity e_UserProfile "User Profile" : Master [
attribute ID Integer [constraints (PrimaryKey)]
attribute email : Email [constraints (NotNull Unique)]
attribute name : String [constraints (NotNull)]
attribute password : Encrypted [constraints (NotNull)]
description "DataEntity to represent user profiles."
]
...

The error identified is:

```

**\*\*Syntax Error\*\***: Missing colon in the attribute definition for `ID` in `DataEntity e\_UserProfile`.

Please provide the corrected RSL specification that is complete and free of the described errors above.

### Prompt for “Semantic Error”:

The following RSL specification contains an error:

```
...
UseCase uc_ManageUserProfile "Manage User Profile": EntitiesManage [
primaryActor a_User
dataEntity e_UserProfile
actions a_CreateProfile, a_EditProfile, a_DeleteProfile,
a_SendConfirmationEmail
precondition "User must be an administrator."
description "Allows users to manage profiles only if they are
administrators, and sends a confirmation email after each action." ]
```

The error identified is:

**\*\*Semantic Error\*\***: In UseCase uc\_ManageUserProfile, an incorrect precondition is added.

Please provide the corrected RSL specification that is complete and free of the described errors above.

### Prompt for “Omission Error”:

The following RSL specification contains an error:

```
...
ActionType a_CreateProfile "Create Profile" [
description "Allows creation of a new user profile." ]

ActionType a_EditProfile "Edit Profile" [
description "Allows editing an existing user profile." ]

ActionType a_DeleteProfile "Delete Profile" [
description "Allows deletion of a user profile." ]

UseCase uc_ManageUserProfile "Manage User Profile": EntitiesManage [
primaryActor a_User
dataEntity e_UserProfile
actions a_CreateProfile, a_EditProfile, a_DeleteProfile
description "Allows users to manage profiles only if they are
administrators, and sends a confirmation email after each action." ]
```

The error identified is:

**\*\*Omission Error\*\***: The `ActionType a\_SendConfirmationEmail` is missing, and it is not included in the `UseCase uc\_ManageUserProfile`.

Please provide the corrected RSL specification that is complete and free of the errors described above.

### Prompt for “Hallucination Error”:

The following RSL specification contains an error:

...

```
Actor a_SupportAgent "Support Agent": Person [
  description "Support agent assisting users." ]
```

```
ActionType a_PasswordRecovery "Password Recovery" [
  description "Allows users to recover forgotten passwords." ]
```

```
UseCase uc_PasswordRecovery "Password Recovery Process": EntitiesManage[
  primaryActor a_User
  secondaryActor a_SupportAgent
  dataEntity e_UserProfile
  actions a_PasswordRecovery
  description "Enables users to recover forgotten passwords with support
  agent assistance." ]
```

The error identified is:

**\*\*Hallucination Error\*\***: It was added Actor a\_SupportAgent, ActionType a\_PasswordRecovery and UseCase uc\_PasswordRecovery, that were not included in the NL requirement.

Please provide the corrected RSL specification that is complete and free of the errors described above.

For each of the errors, the GenAI4RSL-generated RSL specification aligned with the one corrected manually in ITLingo IDE.

### 1.3. Data knowledge

The next step was to improve the model with the corrected dataset to refine its understanding. For this goal, the complete and error-free RSL specification was saved in the GENAI4RSL knowledge base.

```
System UserManagement "User Profile Management": Application [
  description "System to manage user profiles including creation, editing,
  deletion, and email notifications." ]
```

```
Actor a_User "User" : User [
  description "End user managing their profile."
]
```

```
DataEntity e_UserProfile "User Profile" : Master [
  attribute ID : Integer [constraints (PrimaryKey)]
```

```

attribute email : Email [constraints (NotNull Unique)]
attribute Name : String [constraints (NotNull)]
attribute password : String [constraints (NotNull)]
description "DataEntity to represent user profiles."
]

ActionType a_CreateProfile "Create Profile" [
  description "Allows creation of a new user profile." ]

ActionType a_EditProfile "Edit Profile" [
  description "Allows editing an existing user profile." ]

ActionType a_DeleteProfile "Delete Profile" [
  description "Allows deletion of a user profile." ]

ActionType a_SendConfirmationEmail "Send Confirmation Email" [
  description "Send a confirmation email after profile actions."
]

UseCase uc_ManageUserProfile "Manage User Profile" : EntitiesManage [
  primaryActor a_User
  dataEntity e_UserProfile
  actions a_CreateProfile, a_EditProfile, a_DeleteProfile,
  a_SendConfirmationEmail
  description "Allows users to create, edit, or delete their profiles,
  and sends a confirmation email after each action." ]

```

The prompting process and correcting the GenAI4RSL results in ITLingo IDE reflect the loop of interactions for translations to RSL specifications in building all the previously described systems. The results were progressively error-free.

## 2. Prompting the request for RSL specifications to be translated to NL and providing feedback on the generated output

Description: Requesting conversion from RSL language back to NL requirement specification.

Prompt:

```

Convert the following RSL language into a natural language requirement:
[SystemType LoginSystem [description "System to allow users to log in
using their email and password"]
Stakeholder stk_User "User" : Person : Person_User [description "An end-
user who interacts with the system to log in"]
Actor a_User "User" : User [
  stakeholder stk_User
  description "An end-user who logs in using their email and password"]
DataEntity e_User "User" : Master [
  attribute email "Email" : Email [constraints (NotNull Unique)]

```

```

    attribute password "Password" : String [constraints (NotNull
Encrypted)]
    description "Stores user credentials for login"]
UseCase uc_UserLogin "User Login" : Login [
    primaryActor a_User
    dataEntity e_User
    actions a_EnterEmail, a_EnterPassword, a_SubmitLogin
    description "Allows a user to log in using their email and password"]
ActionType a_EnterEmail "Enter Email" [description "User enters their
email"]
ActionType a_EnterPassword "Enter Password" [description "User enters
their password"]
ActionType a_SubmitLogin "Submit Login" [description "User submits login
form"]
]

```

The first translation resulted in the following specifications with omissions:

The system, referred to as "LoginSystem," enables users to log in using their email and password. The primary stakeholder is the "User," defined as an end-user who interacts with the system to log in. The actor "User" is associated with this stakeholder and is described as an end-user who logs in using their email and password. The system stores user credentials in the "User" data entity, which includes attributes for email and password. The email attribute must be unique and not null, while the password attribute must be encrypted and not null. The "User Login" use case allows a user to log in using their email and password, involving actions to enter the email, enter the password, and submit the login form. The actions are defined as follows: "Enter Email" for entering the email, "Enter Password" for entering the password, and "Submit Login" for submitting the login form.

Explanation: The description of the "User" data entity omits the requirement that the email attribute must be unique.

Another translation to NL resulted in the following hallucination:

The system, referred to as "LoginSystem," is designed to allow users to log in using their email and password. The primary stakeholder is the "User," an individual who interacts with the system to perform login operations. The actor "User" is associated with this stakeholder and is responsible for entering their credentials. The system securely stores user credentials in a "User" data entity, which includes the attributes email and password. The email attribute must be unique and not null, while the password attribute must be encrypted and not null. Additionally, the system automatically sends a verification email upon successful login to ensure security. The "User Login" use case defines actions for entering the email, entering the password, and submitting the login form.

Explanation: The feature "Additionally, the system automatically sends a

verification email upon successful login to ensure security” is not part of the original RSL specification. The original specification does not mention any functionality related to sending verification emails upon login.

To correct the described errors the following prompts were elaborated:

- **Omission**

You are an expert in converting RSL (Requirements Specification Language) into accurate natural language requirements.  
Given the following RSL specification:  
[Insert RSL specification here],  
Review the generated natural language description for accuracy. If any information from the RSL specification is missing or omitted in the response, identify and correct the omission. Then, provide a complete and accurate revised version that includes all details from the RSL specification.  
Focus on ensuring that every attribute, constraint, and description in the RSL specification is reflected in the final response.

- **Hallucination**

You are an expert in converting RSL (Requirements Specification Language) into accurate natural language requirements.  
Given the following RSL specification:  
[Insert RSL specification here],  
Review the generated natural language description for accuracy. If any information from the RSL specification is missing, omitted, or included in the response but not in the RSL specification, identify and correct the omission or hallucination. Then, provide a complete and accurate revised version that includes all details and no more from the RSL specification.  
Focus on ensuring that every attribute, constraint, and description in the RSL specification is reflected in the final response.

After the prompt refinement, the result was a complete and accurate translation from RSL specification to NL.

**Output:**

The system must provide a secure login functionality that allows users to access their accounts by entering their registered email address and password. The login interface should prompt users to input their email first, followed by their password, ensuring both fields are completed before proceeding. Once the credentials are submitted, the system should validate the information against the stored user data. If the login details are correct, the user will be granted access to their account; otherwise, an error message will be displayed, prompting the user to retry.

Explanation: This NL requirement specification represents the functionality defined by the RSL translation.

## 2.1 Resume in Initial Conversions and Error Corrections

The first results obtained resulted in several syntax and semantic errors, omissions, and hallucinations in the process of translating NL requirements into RSL specifications and back to NL. In summary, the solutions were described in the previous points, by prompting and requesting the GenAI4RSL to correct them:

- **Syntax Errors:** When syntax errors were detected (e.g., missing colons or incorrect attribute definitions), the prompts were refined to explicitly highlight correct RSL syntax, requiring the model to identify and correct the specified syntax issues. These modifications included direct instructions to fix the exact error and produce a syntactically valid RSL specification.
- **Semantic Errors:** For semantic discrepancies (e.g., misinterpreting requirements or adding incorrect preconditions), the prompts were adjusted to provide more contextual guidance. This involved emphasizing the need to maintain alignment with the original NL requirements and explicitly instructing the model to verify that all conditions accurately reflect the intended meaning without introducing contradictions.
- **Omission Errors:** When elements specified in the NL requirements were omitted in the RSL output, the prompts were enhanced to direct the model to double-check for missing constructs. The revised prompts requested a complete and error-free specification, encouraging the model to review the given requirements and ensure the inclusion of all stated elements before finalizing the output.
- **Hallucination Errors:** In cases where the model introduced extraneous elements not present in the original requirement, the prompts were refined to emphasize strict adherence to the source specification. These prompt modifications reinforced the instruction that no additional, unrequested actors, actions, or other constructs should

appear in the translated RSL output, thus reducing the likelihood of hallucinated content.

Additionally, as illustrated by the examples above, the output from translating RSL specifications back to NL requirements frequently lacked structure, failing to clearly distinguish functional requirements and specify all necessary elements.

This inconsistency triggered the inclusion of a structure in GenAI4RSL instructions for this type of translation. Additional details will be given in the next section regarding prompting and requirement specification in NL.

#### **4.4 Iterative Prompt Engineering & Feedback**

To improve GenAI4RSL's accuracy, it is essential to implement appropriate training methodologies. This training adopts a strategic approach known as iterative prompt engineering, demonstrated during the initial training phase, which involves refining the chatbot's interpretation and response mechanisms to user inputs [5]. This technique is essential for optimizing GenAI4RSL's performance, particularly in translating complex software requirements into the formalized language of RSL.

The process begins with the initial set of prompts—predefined inputs designed to elicit specific responses from the chatbot. These prompts are crafted based on typical scenarios encountered in software requirements specification tasks. Each prompt is analyzed to ensure it encapsulates essential elements that accurately represent the complexity and diversity of natural language expressions used in software development projects.

As the GenAI4RSL interacts with these prompts, its responses are recorded and evaluated against a set of expected outcomes, mainly by validating the generated RSL in the IT-Lingo IDE. This evaluation phase is crucial as it highlights discrepancies and inaccuracies in the chatbot's understanding and translation capabilities. Feedback from this evaluation phase was systematically incorporated into the training cycle by integrating reviewed and corrected versions of the RSL translations, thus enhancing the GenAI4RSL outputs.

In the initial datasets, such as the User Management System, the prompt engineering process was presented in a detailed, step-by-step manner to demonstrate how stable and effective prompts were developed (see Appendix C) and evaluated (see Appendix D). As the

research progressed and these understandings led to more consistent and systematic constructs within the RSL framework, subsequent systems were documented using only the final, integrated prompts. This approach reflects the maturation of the methodology, as early experimentation and incremental refinement ultimately rendered exhaustive intermediate detail unnecessary.

Summary of the iterative prompt engineering approach, as exemplified previously in initial training:

1. **Initial Prompts:** Designed based on typical SRS scenarios.
2. **Response Evaluation:** Assessed the chatbot's outputs against expected results, identifying areas of improvement.
3. **Feedback Integration:** Adjusted prompts and updated the knowledge base with corrected examples.
4. **Refinement:** Repeated the cycle to enhance performance.

The same process was applied in RSL specifications translations to NL, as described in the initial training phase. The prompting intended to encapsulate an output that could be consistent and systematized, to obtain the same structure of requirements adjusted to each specification from RSL. For this purpose, it was included in GenAI4RSL instructions, a set of rules; so the output could be in a structured format ensuring that all critical aspects of the requirements are systematically captured and communicated.

The following sections outline the structure used in the translation from the RSL specification to the NL process:

*Table 4.5 - Software Requirements Specification Structure in NL*

<b>Section</b>	<b>Description</b>
<b>1. General Vision</b>	Provides a concise summary of the specified requirement, offering a high-level overview of the intended functionality or goal.
<b>2. Glossary</b>	States the main terms used throughout the requirement document. This glossary ensures that all stakeholders have a shared understanding of the terminology, ensuring consistency.

<b>3. Stakeholders</b>	Identifies and describes the stakeholders involved in the project, as specified in the RSL language. Outline who has an interest in the requirement and their roles or concerns.
<b>4. Actors</b>	Lists and describes the actors involved in the requirement. Actors refer to entities (such as users or external systems) that interact with the system to achieve a specific goal.
<b>5. Data Entities</b>	Details the data entities and data enumerations described in the RSL language. Data entities refer to objects or concepts relevant to the system's operations, manipulated by actors.
<b>6. Data Entity Clusters</b>	States the data entity clusters, grouping related data entities. This clustering organizes data entities to reflect their relationships and interactions within the system.
<b>7. Functional Requirements</b>	Covers core functionalities described in the RSL language, including use cases, state machines, and actions.
<b>7.1 Description</b>	A detailed description of the use case, as outlined in the RSL language.
<b>7.2 Main Actor</b>	Identification of the primary actor involved in the use case.
<b>7.3 DataEntities</b>	A listing of the data entities relevant to the use case.
<b>7.4 Actions</b>	A breakdown of the actions performed within the use case.
<b>7.5 Where and How</b>	Describes where and how the use case will be executed within the application or software environment.
<b>8. Non-Functional Requirements</b>	Derives non-functional requirements from the functional requirements described earlier. Refers to quality attributes such as performance, security, and usability.
<b>9. Menu Structure and Tabs</b>	Describes the menu structures and tabs within the application related to the functional requirements, helping to understand the affected user interface elements.

This structured approach ensures that all essential elements are covered, promoting completeness and facilitating easier navigation and comprehension.

Following the development of the UserManagementSystem, BillingSystem, LibraryManagementSystem, and AppointmentSchedulerSystem, and their inclusion as part of the knowledge base, combined with feedback integration and prompt refinement, the chatbot progressively evolved into a more sophisticated tool capable of addressing the complexities of software requirements specification. This iterative prompt engineering approach not only improved the chatbot's ability to accurately convert natural language (NL)

requirements into RSL specifications and vice versa but also ensured its adaptability to the evolving landscapes of software development practices and user needs.

While the results became more consistent in both translations, some instances of errors generated by the GenAI4RSL were observed. These issues will be further analyzed and detailed in the Evaluation stage.

# 5

## **Demonstration**

## 5 Demonstration

The Demonstration phase serves as a crucial assessment of GenAI4RSL's capabilities within real-world software engineering scenarios. In this phase, the system's functionality—specifically, its ability to translate natural language (NL) requirements into Requirements Specification Language (RSL), and subsequently retranslate RSL specifications back into NL—is examined. The overarching objective is to assess how closely GenAI4RSL aligns with established software specification practices and to identify areas requiring iterative refinement.

To situate the evaluation in a realistic industrial setting, the selected test case centers on a managing accounting project module. The initial requirements, originally expressed in NL, were input into GenAI4RSL to produce an RSL-based specification. These RSL outputs were then validated using the ITLingo IDE to ensure their syntactic correctness and adherence to standardized specification formats. After validation and refinement, a consolidated RSL file was produced. This process demonstrates the system's potential for seamless integration into typical requirements engineering workflows while simultaneously highlighting opportunities for further enhancement.

The accounting module under examination facilitates the integration of revenue articles from a billing system into an accounting management system for classification and reporting purposes. This integration follows a synchronization process designed to optimize data handling and ensure accurate propagation of revenue information throughout related financial processes. The module addresses the following functionalities:

1. **SQL Coding Task:** Defines SQL structures to generate and aggregate a list of revenue articles.
2. **Data Storage and Interface Creation:** Establishes DataEntities and associated attributes to store revenue articles and create a front-end interface for displaying these articles and their accounting classifications.

3. **Classification and Editing Capabilities:** Supports the classification of revenue articles as income or costs, allows editing, and ensures the maintenance of associations with the accounting management structure.
4. **ERP Synchronization:** Facilitates data flow between the ERP and the accounting management system. This includes running SQL code to update and synchronize records.
5. **Automatic Record Generation:** Automatically generates accounting records within the system by linking synchronized data to defined classifications.
6. **Reporting Integration:** Associate revenue articles with unit types to support detailed reporting on products or services.

To clearly illustrate how these functionalities align with both business processes and technical architectures, a mapping was performed that bridges the conceptual level (functionalities) and the application level (usage patterns). This view ensures a cohesive understanding of how each capability supports broader organizational objectives.

Mapping the abstract level from the described functionalities, we obtained the following architecture:

1. **Application architecture in the viewpoint of application usage**

**Description:** The billing system, integrated into a municipal public revenue framework, produces invoices associated with revenue articles. Each invoice triggers the generation of a corresponding financial record in the Financial Accounting System. A process termed “Application\_connector\_OLD” synchronizes the revenue articles and their financial records into the Accounting Management System, thus automating record generation and improving data consistency across the ecosystem.

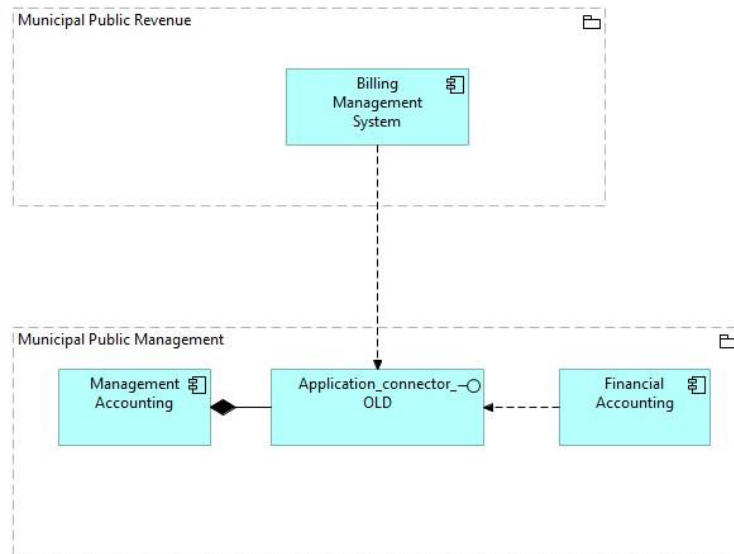


Figure 5.6 - Application Architecture (Application Usage)

The NL requirements detailing these functionalities were translated into RSL using GenAI4RSL. Table 5.6 summarizes how each requirement maps to RSL constructs, including DataEntities, UseCases, and related notes:

Table 5.6 - Requirements translation to RSL

Requirement	DataEntity Definitions	Use Cases	Notes
<b>1. SQL Coding Task</b>	e_RevenueItems, e_RevenueItemsClassification, e_RevenueItemsDistribution	N/A	Manages revenue article data within the accounting management structure.
<b>2. Data Storage and Interface Creation</b>	e_RevenueItems, e_RevenueItemsClassification	uc_1_RevenueItemsImport	Describes the structure of tables and attributes for storing article data.
<b>3. Classification and Editing Capabilities</b>	e_RevenueItemsDistribution	uc_1_2_RevenueItemsImport, uc_1_3_AutomateAccountingMovements	Manages distribution of articles linked to organizational units and activities

			through foreign keys.
<b>4. ERP Synchronization</b>	Implied via events and use cases	uc_1_3_AutomateAccountingMovements	Manages data flow and updates across the system, linking financial accounts with classified and distributed articles.
<b>5. Automatic Record Generation</b>	Related data entities and clusters	uc_1_3_AutomateAccountingMovements	Automates movements in the accounting system, connecting classifications and distributions, including the creation and updating of records.
<b>6. Reporting Integration</b>	e_UnitTypes	uc_UnitTypes	Allows the system to manage and report on article types linked to units of production or service.

The complete RSL specification is presented in Appendix D5.

The next step involved reconvertng the validated RSL specifications into a well-structured, NL-based requirements set using GenAI4RSL. The objective was to produce a specification adhering to ISO/IEC/IEEE 29148 standards, ensuring clarity, completeness, and stakeholder alignment. These standards emphasize maintaining clear, consistent requirements that are easily reviewed and validated by all involved parties. This structured NL output supports the standard’s guidelines by:

- Distinguishing functional from non-functional requirements.

- Facilitating comprehension and verification by stakeholders with diverse technical backgrounds.
- Providing a well-defined format for subsequent lifecycle stages, from design to maintenance.

The ISO/IEC/IEEE 29148:2018 standard, titled "Systems and software engineering—Life cycle processes—Requirements engineering," provides comprehensive guidelines for developing high-quality software requirements. It emphasizes the importance of clear, complete, consistent, and verifiable requirements that are essential for successful software development projects [27].

The structured natural language requirements are designed to be easily reviewed and validated by stakeholders, including those without technical backgrounds. This supports the standard's emphasis on involving stakeholders throughout the requirements engineering process to ensure that the system meets their needs and expectations.

The approach supports the integration of requirements engineering activities within the broader system and software life cycle processes, as outlined in the standard. By providing clear and well-structured requirements, it will facilitate subsequent phases such as design, implementation, testing, and maintenance.

Also, by translating rigorous RSL specifications into structured NL, the objective is to maintain the precision of rigorous methods. The controlled language aligns with the standard's recommendation to use language that is:

- **Simple and Clear:** Facilitates understanding among diverse stakeholders.
- **Consistent:** Avoids synonyms and varied expressions for the same concepts.
- **Free of Jargon:** Minimizes technical terms that may not be universally understood.

By adhering to ISO/IEC/IEEE 29148, the intent is to align the work with internationally recognized best practices in requirements engineering. This enhances the credibility and professionalism of the requirements documentation, making it suitable for use in industry and academic settings [27].

In the initial attempts, GenAI4RSL failed to include all the DataEntities, specified incorrect UseCases, and generated improper StateMachine syntax from the RSL language, resulting in incomplete or incorrect requirements. After manually correcting these issues and

applying the techniques previously described in **Initial Training** from the section **The GenAI4RSL chatbot**, the GenAI4RSL was able to systematically produce the requirements, comprehensively representing the majority of the elements as defined by the RSL language.

The refined requirements in NL iterated upon, are presented in Appendix D5.

## 5.1 Develop Test Instructions

A structured set of test instructions was established to evaluate GenAI4RSL's performance thoroughly. The steps included:

1. Converting NL specifications into RSL;
2. Comparing GenAI4RSL's RSL output against the manually refined RSL specifications;
3. Converting RSL specifications back into NL;
4. Comparing GenAI4RSL's final NL outputs against the manually refined NL documents.

These evaluations were conducted with thirteen professionals—developers, DevOps, testers, and project managers—ensuring diverse expertise and comprehensive feedback.

## 5.2 Test Execution

Test execution involved providing the evaluators with access to GenAI4RSL. Since GPT-4 allows sharing chatbots, evaluators could directly interact with the system using carefully designed prompts. These prompts instructed GenAI4RSL to perform the conversions in alignment with the specifications outlined in Appendix D5.

For the conversion from NL to RSL, the objective is for the GenAI4RSL to produce the following elements: Actors, DataEntities, UseCases, Actions, and StateMachine, as described in the original requirements.

For the conversion from RSL to NL, the goal is for the GenAI4RSL to generate requirements in a use-case style aligned with ISO/IEC/IEEE 29148.

For the first step (NL to RSL), the prompt used was the following:

"Given the set of requirements in natural language (NL) presented below, define these requirements in RSL language, ensuring the specification of the following types of elements: Stakeholders, Actors, DataEntities, UseCases, Actions, and StateMachine.

[Insert the content of the document "NL1\_CG\_Original Specification\_Requirements\_Revenue\_Items" here]"

For the second instruction (RSL to NL), the prompt used was the following:

"Given the set of requirements in RSL language presented below, define these requirements in natural language, ensuring the specification of the following types of elements: Actors, DataEntities, UseCases, Actions, and StateMachine. This conversion should consider ISO/IEC/IEEE 29148 standards and describe the functional requirements in the use case style and the non-functional requirements (if identified).

[Insert the content of the generated RSL code (RSL1) here]"

Through this evaluation, the demonstration phase aims to provide insights into GenAI4RSL's accuracy, reliability, and areas that may require further refinement. By identifying strengths and limitations during this demonstration, subsequent improvements can be more precisely targeted, ultimately enhancing the system's utility in real-world requirements engineering contexts.

# 6

## **Evaluation**

## **6 Evaluation**

The evaluation criteria will include the accuracy of converting natural language to RSL, time in the specification, requirement consistency and user satisfaction, and incorporating feedback for continuous improvement. The process will employ analytical methods, such as comparing model-generated specifications against industry quality standards; observational methods, such as case studies in real-scenario projects; and empirical methods, primarily surveys and interviews with evaluators to measure satisfaction and usability. Data analysis will involve qualitative assessments, such as thematic analysis of interviews to discern patterns, and quantitative assessments, primarily statistical analysis of surveys, focusing on response times and accuracy.

The feedback obtained from accuracy evaluations, time efficiency analyses, and user satisfaction surveys will be systematically compiled and analyzed. The process will involve categorizing feedback into manageable thematic areas such as usability, conversion process, conversion accuracy, and the impact of AI in using requirements engineering. Prioritization will be based on the potential impact of each issue on the model's accuracy and the overall user experience. The aim of this stage is to substantiate the model's accuracy and user satisfaction.

### **Questionnaire Design**

The questionnaire employed in this study was designed to capture a comprehensive understanding of participants' perceptions, attitudes, and experiences related to the impact of AI in software engineering. The instrument comprised 30 questions, structured to ensure both depth and breadth in data collection. For the evaluation of the experience, a questionnaire was built. This document aims to evaluate the use of a chatbot, named GenAI4RSL, in the process of converting requirements specified in NL to RSL and back to NL requirements. It seeks to understand the effectiveness of GenAI4RSL, the clarity and

complexity of the described process, as well as the accuracy of the conversions performed in accordance with the ISO/IEC/IEEE 29148 standard.

### **Likert Scale Questions**

Seventeen of the questions utilized a 5-point Likert scale, ranging from "Strongly Disagree" (1) to "Strongly Agree" (5). This structure was adopted to measure the intensity of participants' attitudes, beliefs, and satisfaction levels regarding specific aspects of the study's core themes. The use of a Likert scale facilitated the quantification of subjective opinions, enabling the conversion of qualitative attitudes into quantitative data. This approach is widely recognized for its effectiveness in capturing the nuances of respondent perceptions while maintaining consistency across responses [52].

The Likert scale questions were categorized into the following thematic areas:

- Evaluation of GenAI4RSL Usage;
- Evaluation of Conversion Processes;
- Evaluation of Conversion Accuracy;
- Evaluation of Requirements Engineering and the Impact of Using Artificial Intelligence;

### **Closed-Ended Questions**

In addition to the Likert scale questions, the questionnaire included ten closed-ended questions. These questions were designed to elicit specific information and to categorize respondents into distinct groups based on demographic and background variables. Examples of closed-ended questions included multiple-choice selections and ranking preferences.

The inclusion of closed-ended questions provided structured data that facilitated easy comparison and statistical analysis, enhancing the reliability of the findings [10]. These questions addressed critical aspects such as:

- Demographic information (e.g., age, gender)
- Experience with the subject matter (e.g., role in the software development process, years of experience in software development, types of projects)
- Evaluation of Requirements Engineering (e.g., frequency of reviewing a requirement specification, consistency in requirements documentation, tools used to detect vague terms in a specified requirement, traceability of software requirements)

### **Open-Ended Questions**

To complement the quantitative data and enrich the overall analysis, three open-ended questions were incorporated. These questions allowed participants to provide detailed, unstructured responses, offering insights into their thoughts, feelings, and suggestions that may not be captured through the closed-ended format. The open-ended questions were designed to explore:

- Participant's general opinions and experiences with the subject matter (e.g., challenges encountered while using the tool, challenges in the conversion process, impact in the software engineering process).
- Suggestions for improvement or feedback on specific aspects (e.g., challenges encountered while using the tool, challenges in the conversion process, impact in the software engineering process).
- Any additional comments or observations considered relevant (e.g., challenges encountered while using the tool, challenges in the conversion process, impact in the software engineering process).

The inclusion of open-ended questions ensured that the study captured a holistic view of participant perspectives, facilitating a more nuanced understanding of the research topic [15].

## **Questionnaire Administration and Validation**

The questionnaire was administered online using Google Forms, ensuring ease of access and convenience for respondents. A pilot study was conducted with a sample of 13 participants to test the clarity, coherence, and reliability of the GenAI4RSL. Based on the feedback received, minor adjustments were made to improve question phrasing and overall flow. The finalized questionnaire was then distributed to the target sample, and responses were collected over a period of [duration, e.g., four weeks].

By combining Likert scale questions with open and closed-ended formats, this questionnaire provided a comprehensive framework for assessing both quantitative and qualitative aspects of the study, contributing to enriching the research objectives.

The expected outcome is a detailed framework of testing, encompassing various dimensions of performance and functionality to ensure a thorough assessment.

## **Interviews**

The semi-structured interview was designed to gain an in-depth understanding of participant's experiences with the GenAI4RSL tool, focusing on their perspectives regarding its usability, functionality, and potential impact on the requirements engineering process. This method was selected due to its flexibility, allowing interviewers to probe deeper into areas of interest while maintaining a core set of questions to ensure consistency across all participants [10].

The interview guide comprised five primary questions, each addressing a distinct aspect of the GenAI4RSL experience. These questions were formulated to align with the study's research objectives, aiming to explore the tool's effectiveness, accuracy, and potential for customization within various roles (e.g., project management, testing, and programming). The questions are as follows:

- 1. General User Experience:**

- *How did using GenAI4RSL impact your workflow during the requirements engineering process? Were there any specific features that enhanced or hindered your efficiency?*
- Rationale: This question aimed to assess the overall usability and integration of the tool within the participant's workflows, identifying key aspects that influenced their experience.

**2. Handling Complex Requirements:**

- *Did you find that GenAI4RSL adequately handled complex or domain-specific requirements? If not, where did you encounter the most difficulties?*
- Rationale: This question sought to evaluate the tool's capability in processing complex requirements, highlighting areas where it may need improvement.

**3. Issues with Output Accuracy:**

- *In your experience, how reliable were the outputs generated by GenAI4RSL when converting natural language to RSL? Were there instances where the outputs were incomplete or inconsistent?*
- Rationale: The goal here was to investigate the accuracy and reliability of the tool's outputs, providing insights into its strengths and limitations in handling natural language inputs.

**4. Customization and Improvements:**

- *What customizations or additional features would you suggest to improve the tool's ability to handle more complex requirements or specific project needs?*
- Rationale: This question encouraged participants to offer suggestions for enhancing the tool's functionality, which is crucial for understanding how the GenAI4RSL could be adapted to meet diverse user requirements.

**5. Impact of AI on Collaboration:**

- *How did GenAI4RSL help you in performing your specific role (e.g., project management, testing, development)? Were the generated outputs useful for your tasks, or did they require additional refinement?*
- Rationale: This question explored the broader impact of the tool on participants' collaboration and role-specific tasks, providing insights into the

potential benefits and challenges of integrating AI into the requirements engineering process.

The interviews were conducted [in person/virtually] and lasted approximately [30-60 minutes] each. The use of open-ended, exploratory questions allowed for a rich exploration of participants' experiences while maintaining enough structure to ensure comparability across interviews [15].

## **6.1 Participants**

Participants were recruited mainly through professional networks. The inclusion criteria were:

- Minimum of 1 year of experience in software development or requirements specification.
- Roles in programming, testing, project management, or DevOps.

The objective was to diversify the experience levels and roles to obtain a comprehensive perspective. Thirteen participants consented to partake in the study, providing a balance of technical and managerial viewpoints across different stages of the software development lifecycle.

The majority of participants had between 4 to 6 years of experience, with the remaining distributed from 1 to 3 years, 7 to 10 years, and over 10 years of experience in software development or requirements specification.

### **Ethical Considerations**

Before participation, all individuals were provided with an information sheet detailing the purpose of the study, procedures, and their rights, including the right to withdraw at any time. Informed consent was obtained from each participant.

Data collected were anonymized to protect participant identity. All records were stored securely, and access was limited to the research team.

## **6.2 Evaluation Criteria**

The evaluation was based on the following set of key criteria for different levels of assessment:

### **1. Evaluation of GenAI4RSL Usage:**

This criterion focused on how easy and intuitive it was for participants to use the tool.

### **2. Evaluation of Conversion Processes:**

This involved assessing how effectively the tool could translate natural language inputs into structured requirements specification language.

### **3. Evaluation of Conversion Accuracy:**

This criterion measured the accuracy of the translations provided by GenAI4RSL compared to manually created requirements.

### **4. Evaluation of Requirements Engineering and the Impact of Using Artificial Intelligence:**

This involved understanding how AI tools like GenAI4RSL could influence the broader process of requirements engineering, including efficiency, consistency, and error reduction.

## **6.3 Survey Analysis**

The methodology followed for evaluating the survey results is based on quantitative and qualitative, interview analysis and respective results. The statistical summary of the quantitative analysis is presented in Appendix E, table 12.15.

## **6.4 Quantitative Analysis**

### **6.4.1 Demographics and Professional Information**

Table 6.7 - Demographic statistics

<i>Category</i>	<i>Subcategory</i>	<i>Number of Respondents</i>	<i>Percentage</i>
<b>Age Distribution</b>	Under 25	1	7.7%
	25-34	6	46.2%
	35-44	2	15.4%
	45-54	4	30.8%
	<b>Total</b>	<b>13</b>	<b>100%</b>
<b>Gender Distribution</b>	Male	9	69%
	Female	4	31%
	<b>Total</b>	<b>13</b>	<b>100%</b>
<b>Role in Software Development</b>	Programmer	7	53.8%
	Tester	3	23.1%
	DevOps	1	7.7%
	Project Manager	2	15.4%
	<b>Total</b>	<b>13</b>	<b>100%</b>
<b>Experience Distribution</b>	Less than 1 year	2	15.4%
	1-3 years	3	23.1%
	4-6 years	3	23.1%
	7-10 years	2	15.4%
	More than 10 years	3	23.1%
	<b>Total</b>	<b>13</b>	<b>100%</b>
<b>Project Type Distribution</b>	Web Development	9	69.2%
	Enterprise Software	3	23.1%
	Mobile Development	1	7.7%
	<b>Total</b>	<b>13</b>	<b>100%</b>

The demographic data reveal a diverse respondent pool, with variations in age, gender, professional roles, and experience levels. Key insights include:

- **Age Distribution:** Most respondents are between 25-34 years (46.2%) and 45-54 years (30.8%).
- **Role Distribution:** A majority are Programmers (53.8%), followed by Testers (23.1%), Project Managers (15.4%), and DevOps (7.7%).
- **Experience and Project Type:** Respondents have varying experience levels, with most engaged in web development (69.2%).

### 6.4.2 GenAI4RSL Usability and Impact

Table 6.8 - Usability and Impact Scores

<i>Aspect</i>	<i>Average Score</i>
<i>Ease of Use</i>	4.46
<i>Clarity of Instructions</i>	3.92

<i>Intuitiveness of Interface</i>	4.31
<i>NL Processing Capability</i>	3.69
<i>Usefulness of Suggestions</i>	3.62

Table 6.8 highlights scores on usability and user experience:

- **Ease of Use:** The tool scored 4.46, with most respondents finding it accessible.
- **Interface Intuitiveness:** Rated at 4.31, indicating that users could navigate the tool effectively.
- **Clarity of Instructions and NLP Capability:** Both scored lower (3.92 and 3.69, respectively), suggesting areas where improvements could enhance user comprehension and language processing.

### 6.4.3 Conversion Accuracy and Process Evaluation

Table 6.9 - Accuracy and Process Evaluation

<i>Aspect</i>	<i>Average Score</i>
<i>NL to RSL Accuracy</i>	3.54
<i>RSL to NL Accuracy</i>	3.77
<i>Logic and Format Preservation</i>	3.69
<i>Inclusion of All Requirements</i>	3.62
<i>Clarity of Requirements</i>	3.85
<i>Consistency of Language</i>	4.23
<i>Organization and Formatting</i>	4.23

Table 6.8 presents findings on conversion accuracy:

- **Consistency and Clarity:** High scores in language consistency (4.23) and clarity (3.85) indicate that the tool maintains a reliable format, beneficial for clear requirements documentation.
- **Conversion Accuracy:** Lower scores for NL to RSL accuracy (3.54) and inclusion of requirements (3.62) highlight a need for refinement in capturing all requirements during conversion.

#### 6.4.4 Correlating GenAI4RSL Performance with User Roles

Table 6.10 - Comparing Process Evaluation and Conversion Accuracy with Respondent's Roles

<i>Questions</i>	<i>DevOps</i>	<i>Project Manager</i>	<i>Programmer</i>	<i>Programmer &amp; Project Manager</i>	<i>Tester</i>	<i>Weighted Mean</i>
<i>GenAI4RSL was easy to use.</i>	5	5	4.57	5	3.67	4.46
<i>Instructions were clear.</i>	4	4	3.86	4	4	3.92
<i>Interface was intuitive.</i>	5	4	4.43	4	4	4.31
<i>Ability to understand NL</i>	5	4	3.43	3	4	3.69
<i>Suggestions were useful.</i>	5	4	3.43	3	3.67	3.62
<i>Followed instructions easily.</i>	5	4	4.71	5	4.67	4.69
<i>Tasks were clearly defined.</i>	5	4	4.86	5	4.67	4.77
<i>Accuracy NL -&gt; RSL.</i>	5	4	3.29	3	3.67	3.54
<i>Accuracy RSL -&gt; NL.</i>	5	4	3.57	3	4	3.77
<i>Logical conversion RSL -&gt; NL.</i>	5	4	3.57	3	3.67	3.69
<i>Complete requirements RSL -&gt; NL.</i>	5	5	3.43	3	3.33	3.62
<i>Sufficient details RSL -&gt; NL.</i>	5	5	2.57	3	3.33	3.15

<i>Clear requirements RSL -&gt; NL.</i>	5	4	4.14	2	3.33	3.85
<i>Consistent language RSL -&gt; NL.</i>	5	4	4.43	4	3.67	4.23
<i>Organized result RSL -&gt; NL.</i>	5	4	4.29	4	4	4.23
<i>Clarity and precision RSL -&gt; NL.</i>	5	3	2.57	2	4	3.08
<i>Positive impact of AI.</i>	5	4	3.86	3	4	3.92

The DevOps professional attributed high scores across all categories, including intuitiveness (5.0) and organization and formatting (5.0). These scores indicate that DevOps professionals found the tool easy to use and effective for their needs.

Project managers also gave high ratings, particularly for ease of use (5.0) and clarity of instructions (4.0). They rated technical aspects such as NL processing capability (4.0) and logic and format preservation (4.0) positively.

Programmers generally provided positive scores, especially for ease of use (4.57) and intuitiveness of interface (4.43). However, their ratings were slightly lower for clarity of instructions (3.86) and inclusion of all requirements (3.43).

One respondent who fulfilled both the roles of Programmer and Project Manager provided a high rating for ease of use (5.0) but gave slightly lower ratings for intuitiveness of interface (4.0), NL processing capability (3.0), and usefulness of suggestions (3.0).

Testers gave moderately positive ratings across most categories. They rated NL processing capability (4.0) and usefulness of suggestions (3.67) relatively high but gave lower scores for clarity of requirements (3.33) and inclusion of all requirements (3.33). This indicates that testers found the tool generally effective in processing natural language but faced challenges in validating the completeness and clarity of the converted requirements.

This analysis clearly shows that developers and testers share a more technical view compared to the rest of the respondents. Both groups highlighted areas where the tool could improve in accurately capturing all requirements and ensuring clarity and consistency, reflecting their focus on technical precision in the software development process.

#### 6.4.5 Average Time Spent by Respondent Role

Table 6.11 - Average Time Spent by Respondent Role

<i>Respondent Role</i>	<i>Average Time Spent (minutes)</i>
<i>DevOps</i>	37.5
<i>Project Manager</i>	65.0
<i>Programmer</i>	46.7
<i>Programmer and Project Manager</i>	37.5
<i>Tester</i>	35.0

#### Additional statistics:

- Average Time Spent: Respondents spent an average of 44.34 minutes on the process.
- Minimum Time: The minimum time spent was 30 minutes, corresponding to the lower time range ("Under 30 min").
- Maximum Time: The maximum time reported was 65 minutes, representing the upper limit for respondents who reported spending "Above 60 min".

Table 6.10 indicates variations in time spent by role:

- **Project Managers** (65 minutes) spend the most time, likely for the reason of the need for thorough review and oversight to ensure alignment with project goals.
- **Programmers** (46.7 minutes) invest time verifying technical details to ensure outputs are implementable, focusing on code and system design.
- **DevOps professionals** (37.5 minutes) focus on alignment with operational needs, requiring less involvement in detailed specifications.
- **Testers** (35 minutes) prioritize testability and clarity, allowing them to review specifications quickly due to their familiarity with validation-focused tasks.

## 6.4.6 Evaluation of Requirements Engineering Practices

Table 6.12 - Summary of Survey Responses on Requirements Engineering Practices

<i>Question</i>	<i>Count</i>	<i>Unique</i>	<i>Top Response</i>	<i>Frequency</i>
<i>How often do you review a requirement specification to verify its completeness?</i>	13	3	After each important update	8
<i>How do you ensure consistency in requirements documentation?</i>	12	3	Manual checks	7
<i>What tools do you use to detect vague terms in a specified requirement?</i>	13	3	Manual review	7
<i>How do you ensure that software requirements are traceable to their sources?</i>	13	2	Requirements management tools (e.g., Jira)	12

Table 6.11, approaches the survey responses on requirements engineering practices, highlighting that:

- **Review Frequency:** Most respondents (8 out of 13) review specifications after significant updates, showing a proactive approach to maintaining documentation completeness.
- **Consistency in Documentation:** Most respondents (7 out of 12) rely on manual checks to ensure consistency. This highlights a potential area for improvement, where automated tools could support or replace manual efforts.
- **Requirement Traceability:** Nearly all respondents (12 out of 13) use requirements management tools, such as Jira, to ensure traceability. Emphasizing the importance of these tools in maintaining connections between requirements and their sources.
- **Detection of Vague Terms:** Most respondents (7 out of 13) use manual reviews to detect vague terms, showing that manual oversight is common and still heavily relied upon in this area.
- **Impact of AI:** The average rating for the positive impact of AI tools on the requirements specification process is 3.92, with a standard deviation of 0.64. This

indicates a generally positive perception, with most respondents agreeing on the useful role of AI in this context.

## **Consolidated Analysis Summary**

### **Usability and User Experience**

- **Ease of Use:** With an average score of 4.46 and dispersion of 0.66, GenAI4RSL was well-received for user-friendliness, especially among DevOps and Project Managers.
- **Interface Intuitiveness:** The tool's intuitiveness (4.31) supports productive use across roles, although Clarity of Instructions (3.92) could be improved for technical users like Programmers and Testers.
- **Natural Language Processing (NLP) and Suggestions:** Scores for NLP (3.69) and the utility of suggestions (3.62) indicate a need for enhanced interpretation of complex language, especially for Testers and Programmers.

### **Conversion Accuracy and Process Evaluation**

- **Consistency and Clarity:** Scores in language consistency (4.23) and clarity (3.85) ensure the specification's usability, which is critical for Developers and Testers.
- **Logic Preservation:** With a score of 3.62, the tool largely preserves logical flow but could be improved for complex requirements.
- **Accuracy in Conversion:** The NL to RSL accuracy (3.54) and inclusion of requirements (3.62) highlight areas for enhancement, indicating occasional missed details during conversion. The RSL to NL accuracy (3.77) was considered acceptable, although fewer times generated inconsistencies.

## **Statistic Review**

The statistical data in Appendix E offers a valuable understanding of user experiences with the GenAI4RSL tool, particularly in terms of usability and performance. The analysis of mean and standard deviation values allows for an assessment of user consensus across various aspects of the tool. Here's a breakdown of the findings:

## 1. Consistency in Positive Ratings:

- **Ease of Use:**
  - Mean: 4.46, Standard Deviation: 0.66
  - Interpretation: Most users found the tool easy to use, with little variability in ratings, indicating broad agreement.
- **Interface Intuitiveness:**
  - Mean: 4.31, Standard Deviation: 0.63
  - Interpretation: Respondents generally found the interface intuitive, with a high level of consensus.
- **Clarity of Instructions:**
  - Mean: 3.92, Standard Deviation: 0.64
  - Interpretation: The tool's instructions were deemed clear by most users, showing consistency regarding clarity.

## 2. Variability in User Experiences:

- **RSL to NL Conversion Clarity and Precision:**
  - Mean: 3.08, Standard Deviation: 1.38
  - Interpretation: Users had mixed opinions on the clarity and precision of the RSL-to-NL conversion, with a high standard deviation indicating significant variability. Some users found the conversion more precise, while others viewed it less favorably.
- **Inclusion of Functional and Non-Functional Requirements in RSL to NL Conversion:**
  - Mean: 3.62, Standard Deviation: 1.12
  - Interpretation: User's perceptions varied on whether the tool included all requirements, suggesting improvements could enhance consistency in capturing comprehensive specifications.
- **Logical and Format Preservation in NL to RSL Conversion:**
  - Mean: 3.69, Standard Deviation: 1.11

- Interpretation: Opinions on maintaining logical flow and format during conversion varied, reflecting differences in user expectations or experiences.
- **Detail Sufficiency in RSL to NL Conversion:**
  - Mean: 3.15, Standard Deviation: 1.07
  - Interpretation: Mixed responses suggest that the tool could improve in providing sufficient detail for users to understand objectives or perform tasks without additional clarifications.

Overall, these findings suggest that while some users find GenAI4RSL effective in certain aspects of the conversion processes, others experience shortcomings. The significant variability points to potential inconsistencies in the tool's performance or differing user expectations, indicating areas for improvement to enhance user satisfaction and reliability. This suggests that future iterations should focus on improving the tool's functional capabilities to ensure a uniform experience across all user groups.

## **6.5 Qualitative Analysis**

The main purpose of qualitative analysis is to interpret the meanings and implications behind the data, providing a deeper understanding of the subject matter. This approach is particularly useful when exploring complex, subjective, or context-dependent aspects of a topic, where numeric data alone may not capture the full picture.

The qualitative analysis was performed on the open-ended responses in the document, focusing on two key questions:

### **1. Challenges Faced in Using the GenAI4RSL Tool**

#### **Response Examples:**

- "Copying the complete results to a document."
- "Encountering errors related to the input format."

#### **Analysis:**

**Technical Challenges:** Respondents reported technical difficulties, such as errors related to input formats and issues with copying results into other documents. These problems highlight areas where the tool's user interface and error handling could be improved.

**Usability Issues:** The challenge of copying results seamlessly into other documents suggests that the tool's usability could be improved to facilitate better integration with existing workflows. Making these outputs more accessible and easier to work with could enhance the tool's overall efficiency.

## 2. Impact of the GenAI4RSL on the Software Engineering Process:

### Response Examples:

- "Automation, reduction of specification times..."
- "Help in understanding how the requirements should be structured..."

### Analysis:

**Automation and Efficiency:** Respondents highlighted the tool's potential to automate tasks and reduce the time required to specify requirements. This feedback reflects the tool's positive impact on efficiency, suggesting that GenAI4RSL could streamline certain stages of the requirements engineering process.

**Clarity and Understanding:** Several respondents noted that the tool aids in understanding and clarifying requirements, which makes it easier for stakeholders to comprehend complex specifications. This points to the tool's potential to improve clarity in documentation, especially for non-technical stakeholders or those less familiar with technical language.

This analysis reveals strengths and areas for improvement in the GenAI4RSL tool, offering valuable inputs for its further development and refinement.

### Consolidated Analysis Summary

- **Technical Difficulties:** Respondents noted issues with input format and copying results, indicating that technical refinements could better integrate the tool with existing workflows.

- **Usability Concerns:** Users suggested that a more refined interface, including options for exporting results directly, would improve usability and integration.
- **Automation and Efficiency:** The tool’s automation capabilities received positive feedback for reducing time on repetitive tasks, allowing users to focus on higher-level work.
- **Clarity and Understanding:** The tool improves the clarity of requirements, making them easier to understand and implement, which is critical for stakeholder alignment.

### **Additional Findings**

- **Usage Feedback:** While GenAI4RSL is generally intuitive, respondents recommended further interface simplifications to support users with varying technical expertise.
- **Conversion Process Feedback:** Participants suggested adding real-time feedback to enhance accuracy and user experience during the conversion process.
- **Accuracy Feedback:** The tool performed well on standard requirements but showed inconsistencies with complex language. Improved accuracy for sophisticated language structures would broaden its applicability.
- **Impact on Requirements Engineering:** Respondents expressed optimism about AI’s potential in requirements engineering but noted that further refinements are needed for reliable integration, particularly for detailed technical roles such as testers and developers.

## **6.6 Interviews Analysis**

In addition to the survey, semi-structured interviews were conducted with most respondents to gain a deeper understanding of their experience with GenAI4RSL.

### **6.6.1 Broader Context of Feedback**

Most respondents viewed GenAI4RSL as a promising tool with the potential to transform requirements engineering by reducing manual effort. However, they noted that the current

conversion process is incomplete, with reliability issues in handling complex requirements. Respondents highlighted that while the tool is helpful, its accuracy in converting intricate specifications needs improvement to fully support requirements engineering processes.

**Challenges for Testers:** Testers, in particular, faced difficulties in comparing RSL-generated requirements to previously refined versions, indicating a need for enhanced precision and alignment with their expectations for testing and validation.

Throughout the interviews, it became clear that some discrepancies in the results produced by GenAI4RSL were caused by the method used for selecting text in the documents. Two respondents "Developers," encountered issues during the initial conversion step. Specifically, the chatbot did not produce any output when manually pasting selected text into the prompt for the requirements specification. However, when they used keyboard shortcuts (CTRL + A or CTRL + T) to select the text, the chatbot successfully generated the RSL conversion.

A "Project Manager" respondent faced a similar issue in the same step. When using manual selection, the output was incomplete, with some Use Cases omitted and the corresponding Actions and Data Entities missing from the RSL conversion. After applying the same keyboard shortcut method, the outcome improved, and all Use Cases from the initial requirements specification were included in the conversion.

Several respondents noted that while the tool saved time, it initially increased their cognitive load, as they had to manually cross-check the outputs to ensure accuracy.

### **6.6.2 Suggestions for Improvement**

Respondents suggested adding configurable options to refine RSL rules and conversion algorithms, which would improve accuracy and usability. Given that GenAI4RSL relies on OpenAI's GPT model, they expressed optimism that future model advancements could enhance its handling of complex requirements, enabling more precise conversions.

### 6.6.3 Impact of AI on Specific Roles

**Project Managers:** Found that GenAI4RSL facilitated a high-level understanding of requirements but noted that outputs required further refinement due to a lack of domain-specific knowledge. This limitation, partially due to intellectual property constraints, underscores the need for access to comprehensive system knowledge for more accurate conversions.

### 6.6.4 General Perception of AI

Although the generality of respondents acknowledged the potential of GenAI4RSL, **Testers** expressed skepticism about fully replacing manual requirements engineering, especially for complex projects. Their feedback underscores the need for high accuracy and customization in handling nuanced requirements, which are essential for validation processes.

### Consolidated Analysis Summary

The interviews yielded valuable qualitative insights into the strengths, limitations, and role-specific experiences with GenAI4RSL. Key themes include:

- **General Perception of GenAI4RSL:** Respondents appreciated the tool's automation potential in requirements engineering, seeing it as a way to reduce manual effort in specifying and reviewing requirements. However, they noted its current limitations in reliably handling complex requirements.
- **Role-Specific Challenges:** Testers and Developers encountered challenges related to precision and technical input methods, with text selection affecting output consistency. Improvements in alignment with the tester's expectations for clear, complete requirements would increase the tool's utility in validation workflows.
- **Technical Limitations and Cognitive Load:** Respondents found the tool sensitive to input methods, influencing output reliability and increasing cognitive load as users

cross-checked for accuracy. Standardized text selection and improvements in handling complex inputs are needed to enhance user experience and usability.

- **Suggestions for Improvement:** Respondents recommended adding configurable RSL rules and conversion algorithms to allow adjustments based on specific needs. They expressed interest in future GPT advancements, which may enhance the tool's performance in complex requirements conversion.
- **Impact of AI on Different Roles:** While Project Managers and Developers were optimistic about the tool's potential, Testers remained cautious, highlighting the tool's current limitations with precision in complex scenarios. These differences suggest that AI adoption in requirements engineering could vary by role, with technical roles requiring greater accuracy and customization.

**General Perception of AI in Requirements Engineering:** Although most respondents recognized AI's potential in this field, they noted varying expectations by role. Project Managers and Developers showed optimism, while Testers emphasized limitations, underscoring the need for role-specific accuracy. With continued improvements in accuracy and usability, GenAI4RSL could become a valuable tool in requirements engineering, contributing to a faster, more reliable software development process.

## 6.7 Discussion of Evaluation

The evaluation results revealed several strengths and weaknesses in the current version of GenAI4RSL, highlighting areas for potential enhancement. These improvements focus on user interaction, accuracy, role-specific adjustments, and broader integration with software development practices.

### 6.7.1 User Interface Enhancements

Feedback from users across various roles indicated the need for a more intuitive and accessible interface. Enhancing the copy/paste functionality and integrating export/upload options could streamline workflows, making it easier to transfer outputs into existing tools and systems. While such features extend beyond OpenAI's standard chatbot interface, their

inclusion would facilitate seamless integration, improving user interaction and reducing manual effort.

### **6.7.2 Improved Conversion Processes**

Testers and developers emphasized the importance of real-time feedback mechanisms to improve input structuring and reduce errors. Incorporating features such as real-time prompts to flag format or syntax issues during the input phase would enhance the accuracy and consistency of generated RSL outputs. These mechanisms could guide users in adhering to syntactic and structural requirements, minimizing rework, and improving overall reliability.

### **6.7.3 Accuracy Enhancements**

The tool's ability to handle complex and nuanced language was a critical area for improvement, particularly for testers and developers. Incorporating a verification check at the end of each conversion would enable the tool to identify and flag issues such as missing requirements, ambiguous language, or deviations from RSL standards. This feature would provide users with an opportunity to review and correct outputs before finalization, significantly improving trust and usability.

### **6.7.4 Incorporating software project architecture**

Integrating knowledge of software project architectures into GenAI4RSL's knowledge base could enhance the precision of generated functional requirements. By using a deeper understanding of architectural contexts, the tool could produce outputs more closely aligned with project-specific needs. This adjustment would address gaps identified by project managers and developers, who emphasized the importance of architectural relevance in requirements engineering.

### 6.7.5 Integration with Existing Tools

Feedback highlighted the importance of integrating GenAI4RSL with existing requirements management and software development lifecycle tools, such as Jira, Confluence, and Trello. Developing APIs or plugins to synchronize requirements automatically with these platforms would improve workflow compatibility. Such integrations would enable users to incorporate GenAI4RSL seamlessly into their established workflows, maximizing its utility and reducing friction.

### 6.7.6 Role-Specific Adjustments

Tailoring the tool to meet the unique needs of various roles—such as developers, testers, and project managers—was identified as a key area for improvement. Introducing a role-specific preferences feature would allow users to customize settings that influence how the tool structures outputs based on their specific tasks. This role-adaptable functionality could improve efficiency and satisfaction by aligning the tool more closely with the diverse demands of different user roles.

### 6.7.7 Broader Impact on the Software Development Industry

With ongoing refinement, GenAI4RSL has the potential to significantly impact the software development industry by addressing several key areas:

- **Automating Repetitive Tasks:** By reducing the time from initial specification to implementation, the tool could increase productivity across projects.
- **Improving Accuracy and Consistency:** Lowering error rates in requirements documentation could reduce rework, saving costs and enhancing software quality.
- **Increasing Accessibility:** Making requirements engineering more accessible to non-technical stakeholders, such as project managers, could democratize the process and result in more inclusive specifications that meet diverse stakeholder needs.

- **Enabling Seamless Collaboration:** Integration with existing requirements platforms would enable streamlined workflows, and collaboration across roles throughout the software development lifecycle.

### **Summary**

The identified improvements underscore GenAI4RSL's transformative potential in the requirements engineering domain. By systematically addressing its identified weaknesses and building on its existing strengths, the tool can significantly improve accuracy, usability, and integration. This can drive industry-wide adoption and foster greater collaboration across teams. These enhancements align with the broader goal of automating and improving the efficiency of software development processes.

# 7

## **Conclusion**

## 7 Conclusion

The advancement of AI has opened new opportunities in software engineering, particularly in the area of requirements specification. This thesis introduces GenAI4RSL, an AI-driven chatbot utilizing GPT-4 and the RSL, designed to enhance the SRS process. The primary aim of this research was to evaluate how AI can improve efficiency, accuracy, and consistency in translating natural language (NL) requirements into structured formats using RSL, and vice versa.

### 7.1 Summary of Research

The challenges inherent in traditional SRS processes, such as the manual effort required to draft clear and comprehensive specifications, often lead to ambiguities, inconsistencies, or omissions that negatively impact the software development lifecycle. Acknowledging these limitations, this research proposed GenAI4RSL, an AI-powered chatbot based on GPT-4 and RSL, designed to automate key aspects of the requirements drafting process. The primary aim was to improve clarity, reduce errors, and accelerate the development lifecycle, aligning with recent research highlighting the benefits of integrating AI into requirements engineering [14], [59].

The development of GenAI4RSL involved rigorous data collection and pre-processing, utilizing datasets containing NL requirements paired with corresponding RSL specifications. The training process employed iterative prompt engineering to fine-tune GPT-4's ability to understand and generate RSL constructs. Through error analysis, common issues such as syntax errors, omissions, and AI hallucinations were identified and addressed through systematic refinements. These methodologies reflect practices advocated in recent studies, which emphasize the importance of role-prompting, explicit instructions, and iterative refinement in optimizing AI model performance [5], [17], [18].

The use of a CNL, like RSL, was pivotal in systematizing requirements at various abstraction levels and specific concerns. RSL's simplified grammar and predefined

vocabulary aligned with ISO/IEC/IEEE 29148 standards, enabling GenAI4RSL to produce specifications with improved clarity and reduced ambiguity [54]. Without RSL's strict linguistic guidelines, deviations in generated outputs due to inherent AI limitations, such as omissions and hallucinations, would have been more pronounced.

To evaluate GenAI4RSL's practical capabilities, the tool was tested on real-world requirements drawn from a managing accounting project. The demonstration included generating RSL specifications from complex NL requirements and validating the tool's outputs. A mixed-methods evaluation approach was employed, combining quantitative surveys, qualitative feedback, and interviews with software professionals in roles, such as developers, testers, project managers, and DevOps engineers.

The evaluation revealed several understandings of the adoption of GenAI4RSL:

Considering **usability and intuitiveness**, participants found the chatbot's interface user-friendly and intuitive, receiving high ratings for accessibility and ease of use.

Regarding **efficiency gains**, GenAI4RSL demonstrated potential in automating parts of the requirements drafting process, reducing the time required, and enabling stakeholders to focus on higher-level design considerations.

Lastly, in **improving clarity and consistency**, the use of RSL helped the tool to standardize requirements documentation, enhancing clarity and minimizing ambiguity.

However, the research also identified areas for improvement:

Considering **accuracy with complex requirements**, the tool occasionally struggled to handle complex requirements, particularly in maintaining completeness and correctness. Additionally, submitting lengthy requirements into prompts adversely affected output accuracy.

Considering **technical limitations**, issues such as AI hallucinations and sensitivity to input formatting reduced output reliability, requiring manual verification and increasing the cognitive load.

Finally, under **role-specific challenges**, testers and developers faced greater difficulty using the tool in opposition to project managers and DevOps professionals, underscoring the need for tailored functionalities to address the specific requirements of different roles.

The findings suggest that while tools like GenAI4RSL hold significant promise for enhancing requirements engineering, their effective integration into existing workflows is essential. By automating repetitive tasks and standardizing requirements documentation, AI-driven tools can reduce manual effort, allowing stakeholders to focus on high-level decision-making and design. However, the early phases of adoption revealed an increased cognitive load due to the need for manual verification, emphasizing the importance of iterative refinements and user feedback [59].

As AI-powered tools evolve, they must address challenges such as accuracy, adaptability to complex requirements, and hallucination reduction. Despite these limitations, the broader implications of AI in requirements engineering are profound. AI has the potential to reduce human error, foster collaboration between technical and non-technical stakeholders, and enhance the overall efficiency of software development processes.

Reflecting on the work of Aithal and Shubhrajyotsna [1] on the use of AI-based GPTs in research methods, they emphasize the importance of researchers exercising critical judgment, ensuring ethical considerations are met, and maintaining a balance between automation and human expertise [1]. This resonates well with the GenAI4RSL experience, where user reviews and validations remain essential in ensuring accuracy and consistency despite the advancements in automation.

## **7.2 Limitations**

Adopting AI-driven methodologies for requirements specification introduces a range of potential challenges. Key among these is the inherent difficulty in capturing the full spectrum of human communication nuances in technical requirements, the substantial volume of high-

quality training data required for effective machine learning, and resistance from stakeholders accustomed to traditional specification practices.

During the iterative process of GenAI4RSL instructing, the current version, GPT-4, produced occasional responses that were irrelevant or outside the scope of the given queries. These responses, often referred to as "hallucinations," are examples of incorrect or misleading outputs generated by AI models. Despite efforts to refine the knowledge base to improve response consistency, the occurrence of hallucinations persisted and significantly impacted the translation process.

This issue is not unique to GenAI4RSL but represents a broader challenge encountered in many large language models (LLMs), as highlighted by Maleki et al. (2022). The inconsistencies primarily arose during the initial conversion phase, where NL requirements were translated into RSL. This phase's errors propagated into the final step, resulting in incomplete or inaccurate NL outputs. Maleki, Padmanabhan, and Dutta identify several root causes for this phenomenon [42]:

- **Data Limitations:** LLMs heavily rely on training data, which may be incomplete, biased, or misaligned with real-world contexts. Hallucinations occur when models generate content that appears plausible based on their training but lacks grounding in actual data or factual knowledge.
- **Predictive Nature of LLMs:** Text generation in LLMs is inherently statistical, as models predict the most likely sequence of words based on prior input. This approach, while effective for many tasks, lacks true semantic understanding, leading to outputs that are plausible-sounding but inaccurate.
- **Interpretation Challenges:** Translating NL requirements into structured formats like RSL introduces additional complexity. Ambiguities in NL inputs can result in misinterpretations that propagate errors throughout the generated output, reducing consistency and completeness.
- **Generalization Errors:** LLMs occasionally infer details or make assumptions not explicitly present in the input data. These hasty generalizations can produce outputs that deviate significantly from the intended meaning or factual accuracy.

While hallucinations highlight current limitations in AI-driven text generation, they also serve as a critical area for future research. Addressing these challenges will require a multidisciplinary approach involving improvements in model architecture, data quality, and interpretive techniques. Despite these issues, AI-driven tools like GenAI4RSL represent significant potential to enhance requirements engineering processes when carefully designed and iteratively refined.

### 7.3 Future Work

The evolution of AI models, such as GPT-4, is pivotal in addressing current challenges, including AI hallucinations and the handling of domain-specific knowledge. To unlock their full potential, future research should focus on the following areas:

1. **Refining AI Models:** Advances in the underlying language models are essential for improving the comprehension of complex requirements and minimizing errors. Enhanced contextual understanding and domain adaptability are particularly critical in ensuring accurate outputs in specialized fields.
2. **Ethical Considerations:** The integration of AI in requirements engineering must prioritize ethical concerns, including safeguarding data privacy, obtaining informed consent from stakeholders, and mitigating biases in AI outputs. These considerations are vital to building trust and ensuring responsible AI deployment.
3. **Expanding Applications to Related Fields:** AI's applicability extends beyond requirements engineering to other areas of software development, such as test automation and quality assurance. For instance, Cunha, S. (2024) introduces an innovative approach to User Acceptance Testing (UAT) by leveraging Generative AI and RSL to automate the generation of critical test deliverables. These include requirements, test cases, and automation scripts, effectively streamlining the testing process and improving efficiency [38].
4. **Continuous User Engagement:** Actively involving end-users in iterative development cycles ensures that AI tools align with practical needs. This participatory approach bridges the gap between theoretical advancements and real-world applicability, fostering tools that are both effective and user-friendly.

In conclusion, AI-driven tools such as GenAI4RSL demonstrate significant potential to streamline workflows and reduce manual effort in software development. This research underscores the opportunities and challenges posed by integrating AI into software requirements engineering, offering several directions for future work. These include refining AI models for improved handling of specialized domains and embedding ethical considerations into AI development processes.

As AI technologies evolve, they are set to play a central role in automating and enhancing the software development lifecycle, making it more efficient, reducing errors, and fostering collaboration across teams. The future of AI in requirements engineering is promising. With continued innovation, iterative refinement, and proactive user engagement, tools like GenAI4RSL are well-positioned to drive a transformative shift toward dynamic and effective software development practices.

This study contributes to the field of software requirements engineering by presenting GenAI4RSL as a proof of concept that highlights AI's potential in drafting and refining requirements specifications. By addressing existing limitations and implementing the proposed recommendations, AI-driven tools can become indispensable in creating more accurate, efficient, and collaborative development environments.

Realizing the full benefits of AI in requirements engineering requires a multidisciplinary approach that combines technical innovation with ethical responsibility and user-centered design principles. As these technologies continue to advance, their thoughtful integration holds the potential to revolutionize the software industry, yielding higher-quality products and enabling more agile, collaborative development processes.

## References

- 1 Aithal, P. S., & Aithal, S. (2023). Use of AI-based GPTs in experimental, empirical, and exploratory research methods. *International Journal of Case Studies in Business, IT, and Education (IJCSBE)*, 7(3), 411-425.
- 2 Ahmad, K. (2021). Human-centric requirements engineering for artificial intelligence software systems. *IEEE 29th International Requirements Engineering Conference (RE)* (pp. 468-473).
- 3 Ahmad, K., Abdelrazek, M., Arora, C., Bano, M., & Grundy, J. (2023). Requirements engineering for artificial intelligence systems: A systematic mapping study. *Information and Software Technology*, 158, 107176.
- 4 Alhaizaey, A., & Al-Mashari, M. (2023). A framework for reviewing and improving non-functional requirements in agile-based requirements. *18th Iberian Conference on Information Systems and Technologies (CISTI)* (pp. 1-7).
- 5 Amatriain, X. (2024). Prompt design and engineering: Introduction and advanced methods. *arXiv preprint*.
- 6 Balasubramaniam, N., Kauppinen, M., Hiekkänen, K., & Kujala, S. (2022). Transparency and explainability of AI systems: Ethical guidelines in practice. In V. Gervasi & A. Vogelsang (Eds.), *Requirements Engineering: Foundation for Software Quality. REFSQ 2022 (Lecture Notes in Computer Science, vol 13216)*. Springer, Cham.
- 7 Barrocas, A. (2022). Information Analysis and Management for Agile Project Management with the ITLingo-Cloud Platform.
- 8 Bellini, V., Cascella, M., Cutugno, F., Russo, M., Lanza, R., Compagnone, C., & Bignami, E. G. (2022). Understanding basic principles of artificial intelligence: A practical guide for intensivists. *Acta Biomedica*, 93(5), e2022297.
- 9 Bousdekis, A., Lepenioti, K., Apostolou, D., & Mentzas, G. (2021). A review of data-driven decision-making methods for industry 4.0 maintenance applications. *Electronics*, 10(7), 828.
- 10 Bryman, A. (2016). *Social Research Methods* (5th ed.). Oxford University Press.
- 11 Budake, R., Bhoite, S., & Kharade, K. (2023). A study of AI-based techniques for requirement analysis in software engineering. *AIP Conference Proceedings*, 2946(1), 050007.
- 12 Casillo, F., Deufemia, V., & Gravino, C. (2022). Detecting privacy requirements from user stories with NLP transfer learning models. 10.48550/arXiv.2202.01035.
- 13 Chen, Y.-P., Nishida, N., Nakayama, H., & Matsumoto, Y. (2024). Recent trends in personalized dialogue generation: A review of datasets, methodologies, and evaluations. University of Tokyo, RIKEN AIP. 10.48550/arXiv.2405.17974.
- 14 Cheng, H., Husen, J. H., Peralta, S. R., Jiang, B., Yoshioka, N., Ubayashi, N., & Washizaki, H. (2024). Generative AI for Requirements Engineering: A Systematic Literature Review.
- 15 Creswell, J. W., & Creswell, J. D. (2017). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches* (5th ed.). SAGE Publications.
- 16 Dalpiaz, F., & Niu, N. (2020). Requirements engineering in the days of artificial intelligence. *IEEE Software*, 37, 7-10.

- 17 Dwivedi, Y. K., Kshetri, N., Hughes, L., Slade, E. L., Jeyaraj, A., Kar, A. K., & Wright, R. (2023). "So what if ChatGPT wrote it?" Multidisciplinary perspectives on opportunities, challenges and implications of generative conversational AI for research, practice and policy. *International Journal of Information Management*, 71, 102642.
- 18 Ekin, S. (2023). Prompt engineering for ChatGPT: A quick guide to techniques, tips, and best practices. TechRxiv
- 19 Frincke, D., Wolber, D., Fisher, G., & Cohen, G. (1992). Requirements specification language (RSL) and supporting tools.
- 20 Gacenga, F., Cater-Steel, A., Toleman, M., & Tan, W. (2012). A proposal and evaluation of a design method in design science research. *Electronic Journal of Business Research Methods*, 10, 89-100.
- 21 Gao, J., Zhuo, H. K., Li, L., & Caotang, Z. C. (2013). Software requirement acquisition method based on probabilistic action-model algorithm.
- 22 Gaspar, J. G. A. H. G. (2012). Analysis of problems in reference requirements specifications.
- 23 Gill, K. D., Raza, A., Zaidi, A. M., & Kiani, M. M. (2014). Semi-automation for ambiguity resolution in open source software requirements. *IEEE 27th Canadian Conference on Electrical and Computer Engineering (CCECE)* (pp. 1-6).
- 24 Gupta, S., Epiphaniou, G., & Maple, C. (2023). AI-augmented usability evaluation framework for software requirements specification.
- 25 Harth, P., Jähde, O., Schneider, S., Horn, N., & Buchkremer, R. (2023). From data to human-readable requirements: Advancing requirements elicitation through language-transformer-enhanced opportunity mining. *Algorithms*, 16(9), 403.
- 26 Hevner, A., March, S., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly*, 28(1), 75-105.
- 27 ISO/IEC/IEEE International Standard - Systems and software engineering -- Life cycle processes -- Requirements engineering, ISO/IEC/IEEE 29148:2018(E), pp.1-94, Nov. 2018, doi: 10.1109/IEEESTD.2018.8559681
- 28 Jahić, B., Guelfi, N., & Ries, B. (2023). SEMKIS-DSL: A domain-specific language to support requirements engineering of datasets and neural network recognition. *Information*, 14(4), 213.
- 29 Kitchenham, B. A. (2012). Systematic review in software engineering: Where we are and where we should be going. *Proceedings of the 2nd International Workshop on Evidential Assessment of Software Technologies (EAST '12)*. Association for Computing Machinery.
- 30 Kitchenham, B., & Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering (Version 2.3). EBSE Technical Report.
- 31 Krishna, M., Gaur, B., Verma, A., & Jalote, P., (2024). Using LLMs in software requirements specifications: An empirical evaluation.
- 32 Kotti, Z., Galanopoulou, R., & Spinellis, D. (2023). Machine learning for software engineering: a tertiary study. *ACM Computing Surveys*, 55(12), 1-39.
- 33 Lano, K., Fang, S., Umar, M. A., & Yassipour-Tehrani, S. (2020). Enhancing model transformation synthesis using natural language processing. *Proceedings of the 23rd*

- ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (pp. 1-10).
- 34 Lepenioti, K., Bousdekis, A., Apostolou, D., & Mentzas, G. (2021). Human-augmented prescriptive analytics with interactive multi-objective reinforcement learning. *IEEE Access*, 9, 100677-100693.
  - 35 Liu, J., Symons, C., & Vatsavai, R. R. (2022). Persona-Based Conversational AI: State of the Art and Challenges.
  - 36 Maalej, W., Pham, R. Y., & Chazette, L. (2023). Tailoring requirements engineering for responsible AI.
  - 37 Madni, A. (2010). Integrating humans with software and systems: Technical challenges and a research agenda. *Systems Engineering*, 13.
  - 38 Melero dos Santos, & Cunha, S. (2025). An approach to improve User Acceptance Testing with Generative AI and RSL based techniques.
  - 39 Méndez Fernández, D., Wagner, S., Kalinowski, M., Felderer, M., Mafra, P., Vetro, A., Conte, T., Christiansson, M.-T., Greer, D., Lassenius, C., Männistö, T., Nayebi, M., Oivo, M., Penzenstadler, B., Pfahl, D., Prikładnicki, R., Ruhe, G., Schekelmann, A., Sen, S., & Wieringa, R. (2016). Naming the pain in requirements engineering: Contemporary problems, causes, and effects in practice. *Empirical Software Engineering*.
  - 40 Merugu, R. R. R., & Chinnam, S. R. (2021). Automated cloud service based quality requirement classification for software requirement specification. *Evolving Intelligence*, 14, 389–394.
  - 41 Moreno, V., Génova, G., Parra, E., & Fraga, A. (2020). Application of machine learning techniques to the flexible assessment and improvement of requirements quality. *Software Quality Journal*, 28.
  - 42 Maleki, N., Padmanabhan, B., & Dutta, K. (2024). AI Hallucinations: A Misnomer Worth Clarifying. *IEEE Conference on Artificial Intelligence (CAI)*, Singapore, pp. 133-138.
  - 43 Nistala, P. V., Rajbhoj, A., & Kulkarni, V. (2022). Towards digitalization of requirements: Generating context-sensitive user stories from diverse specifications. *Automated Software Engineering*, 29, 26.
  - 44 Parnas, D. L. (1972). A technique for software module specification with examples. *Communications of the ACM*, 15(5), 330-336.
  - 45 Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, 24(3), 45-77.
  - 46 Peng, B., Li, C., He, P., Galley, M., & Gao, J. (2023). Instruction tuning with GPT-4. [10.48550/arXiv.2304.03277](https://arxiv.org/abs/2304.03277).
  - 47 Piorkowski, D., Park, S., Wang, A., Wang, D., Muller, M., & Portnoy, F. (2021). How AI developers overcome communication challenges in a multidisciplinary team. *Proceedings of the ACM on Human-Computer Interaction*, 5, 1-25.
  - 48 Piazza, P., Checcucci, E., Puliatti, S., Belençhòn, I. R., Veccia, A., Rivas, J. G., Taratkin, M., Kowalewski, K.-F., Rodler, S., & Cacciamani, G. E. (2023). The long but necessary journey towards optimization of the cause-effect relationship between input and output for accountable use of ChatGPT for academic purposes.

- 49 Qaisi, H. A., Quba, G. Y., Althunibat, A., Abdallah, A., & Alzu'bi, S. (2021). An intelligent prototype for requirements validation process using machine-learning algorithms. *International Conference on Information Technology* (pp. 870-875).
- 50 Rahimi, N., Eassa, F., & Elrefaei, L. (2020). An ensemble machine learning technique for functional requirement classification. *Symmetry*, 12(10), 1601.
- 51 Ramamoorthy, P., Wei-Tek Tsai, & Usuda (1984). *Software engineering: Problems and perspectives*. *Computer*, 17(10), 191-209.
- 52 Robinson, S. B., & Leonard, K. F. (2018). *Designing quality survey questions*. SAGE Publications.
- 53 Shi, F., et al. (2020). Recent progress on the convergence of the Internet of Things and Artificial Intelligence. *IEEE Network*, 34(5), 8-15.
- 54 Silva, A., & Savic, D. (2021). Linguistic Patterns and Linguistic Styles for Requirements Specification: Focus on Data Entities. *Applied Sciences*, 11, 4119.
- 55 Silva, A. (2019). Rigorous Specification of Use Cases with the RSL Language. *28th International Conference on Information Systems Development (ISD2019)*, Toulon, France.
- 56 Silva, A. R., Savić, D., Vlajić, S., Antović, I., Lazarević, S., Stanojević, V., & Milić, M. (2015). A pattern language for use cases specification. In *Proceedings of EuroPLOP 2015* (pp. 1–18). ACM.
- 57 Sudhi, V., Kutty, L., & Gröpler, R. (2023). Natural language processing for requirements formalization: How to derive new approaches? B. H. Schlingloff, T. Vogel, & A. Skowron (Eds.), *Concurrency, Specification, and Programming* (Vol. 1091, *Studies in Computational Intelligence*, pp. 1-10). Springer, Cham.
- 58 Talele, P., Apte, S., Phalnikar, R., & Talele, H. (2023). Semi-automated software requirements categorization using machine-learning algorithms. *International Journal of Electrical and Computer Engineering Systems*, 14(10), 1107-1114.
- 59 Umar, M. A., & Lano, K. (2024). Advances in automated support for requirements engineering: A systematic literature review. *Requirements Engineering*, 29(2), 177–207.
- 60 Wagner, S., Méndez Fernández, D., Felderer, M., & Kalinowski, M. (2016). Requirements engineering practice and problems in agile projects: Results from an international survey.
- 61 Wang, G., & Xu, J. (2021). Autonomous software requirement specifications towards AI programming. *IEEE 20th International Conference on Cognitive Informatics & Cognitive Computing* (pp. 123-130).
- 62 Wang, Z., Pan, J.-S., Chen, Q., & Yang, S. (2022). BiLSTM-CRF-KG: A construction method of software requirements specification graph. *Applied Sciences*, 12(12), 6016.
- 63 Wangsa, K., Karim, S., Gide, E., & Elkhodr, M., (2024). A Systematic Review and Comprehensive Analysis of Pioneering AI Chatbot Models from Education to Healthcare: ChatGPT, Bard, Llama, Ernie and Grok. *Future Internet*, 16(7), 219.
- 64 Yang, J., & Liu, L. (2020). What users think about predictive analytics? A survey on NFRs. *IEEE 28th International Requirements Engineering Conference (RE)* (pp. 340-345).

# A

## **Appendix A. Schedule**

## Appendix A

DSR Phase	Task	Period
<b>#1</b>	<b>Problem identification and motivation</b>	
	Analyse of the research context and motivation.	Dec/2023
	Understand and analyze the background.	Dec/2023
<b>#2</b>	<b>Define the objectives of a solution</b>	
	Analyze the Related Work: involved technologies.	Jan/2024
	Analyze the Related Work: related projects.	Feb/2024
	Define the research objectives.	Feb/2024
<b>#3</b>	<b>Design and development</b>	
	Iteration-1: Data Collection & Pre-processing	Mar/2024
	Iteration-2: Initial Training	Apr/2024
	Iteration-3: Iterative Prompt Engineering & Feedback	May/2024
<b>#3</b>	<b>Demonstration</b>	
	Iteration-1: Develop Test Criteria	May/2024
	Iteration-2: Execute Tests	Jun/2024
	Iteration-3: Analyse Test Results	Jul/2024
<b>#4</b>	<b>Evaluation</b>	
	Iteration-1: Accuracy Evaluation	Ago/2024
	Iteration-2: Time Efficiency Analysis	Ago/2024
	Iteration-3: User Satisfaction Survey	Ago/2024
<b>#5</b>	<b>Communication</b>	
	Write the research project report.	Jan/2024
	Present and discuss the research project report	Mar/2024
	Write the MSc dissertation.	Jun-Dec/2024
	Present and discuss the MSc dissertation	Update

# B

## **Appendix B. Studies selected in the SLR**

## Appendix B

Table 10.13 - List of Studies included in analysis

ID	Articles	Authors	Year	Type of Publication
S1	"A Framework for Reviewing and Improving Non-Functional Requirements in Agile-based Requirements"	Abdulrahim Alhaizaey, Majed Al-Mashari	2023	Conference
S2	"A Study of AI-based Techniques for Requirement Analysis in Software Engineering"	Rajendra Budake, Sudhakar Bhoite, Kabir Kharade	2022	Conference
S3	"AI-augmented usability evaluation framework for software requirements specification in cyber-physical human systems"	Gupta, Sandeep & Epiphaniou, Gregory & Maple, Carsten	2023	Article
S4	"An Intelligent Prototype for Requirements Validation Process Using Machine Learning Algorithms"	Hadeel Al Qaisi, Gaith Y Quba, Ahmad Althunibat, Ayman Abdallah, Shadi Alzu'bi	2021	Conference
S5	"From Data to Human-Readable Requirements: Advancing Requirements Elicitation through Language-Transformer-Enhanced Opportunity Mining"	Harth P, Jähde O, Schneider S, Horn N, Buchkremer R.	2023	Article
S6	"Natural Language Processing for Requirements Formalization: How to Derive New Approaches?"	Sudhi, V., Kutty, L., Gröpler, R.	2023	Paper
S7	"Software requirement acquisition method based on probabilistic action-model algorithm"	Gao, J., Zhuo, H. K., Li, L., & Caotang, Z. C	2013	Conference
S8	"AI-augmented Model-Based Capabilities in the AIDOaRt Project"	R Raja Ramesh Merugu, Satyananda Reddy Chinnam	2022	Article
S9	"An Ensemble Machine Learning Technique for Functional Requirement Classification"	Rahimi N, Eassa F, Elrefaei L.	2020	Article

<b>S10</b>	“Application of machine learning techniques to the flexible assessment and improvement of requirements quality”	Moreno, Valentín & Génova, Gonzalo & Parra, Eugenio & Fraga, Anabel	2020	Article
<b>S11</b>	“Automated Cloud Service-Based Quality Requirement Classification for Software Requirement Specification”	Merugu, R.R.R., Chinnam, S.R.	2021	Article
<b>S12</b>	“Detecting Privacy Requirements from User Stories with NLP Transfer Learning Models”	Francesco Casillo, Vincenzo Deufemia, Carmine Gravino	2022	Article
<b>S13</b>	“Recent Progress on the Convergence of the Internet of Things and Artificial Intelligence”	Feifei Shi, Huansheng Ning, Wei Huangfu, Fan Zhang, Dawei Wei, Tao Hong, Mahmoud Daneshmand	2020	Article
<b>S14</b>	“Requirements Engineering for Artificial Intelligence Systems: A Systematic Mapping Study”	Khlood Ahmad, Mohamed Abdelrazek, Chetan Arora, Muneera Bano, John Grundy	2023	Review
<b>S15</b>	“SEMKIS-DSL: A Domain-Specific Language to Support Requirements Engineering of Datasets and Neural Network Recognition”	Jahić Benjamin, Guelfi Nicolas, Ries Benoît,	2023	Review
<b>S16</b>	“Towards digitalization of requirements: generating context-sensitive user stories from diverse specifications”	Padmalata V. Nistala, Asha Rajbhoj, Vinay Kulkarni, Shivani Soni, Kesav V. Nori, Raghu Reddy	2022	Article
<b>S17</b>	“Transparency and Explainability of AI Systems: From Ethical Guidelines to Requirements”	Nagadivya Balasubramani, Marjo Kauppinen, Antti Rannisto, Kari Hiekkänen, Sari Kujala	2023	Article

<b>S18</b>	“BiLSTM-CRF-KG: A Construction Method of Software Requirements Specification Graph”	Wang Zhengdia, Pan Jeng-Shyangb, Chen Qinyinc, Yang Shuangyuan	2022	Article
------------	---	--	------	---------

*Table 10.14 - Correlation of Technologies with Requirements by Study*

<b>ID</b>	<b>Technologies</b>	<b>Requirements</b>
<b>S1</b>	Natural Language Processing (NLP), Artificial Intelligence (AI), Machine Learning (ML), Deep Learning (DL)	The framework utilizes NLP and AI to automate analyzing and predicting non-functional requirements in user stories, generating artifacts for requirements inspection.
<b>S2</b>	Artificial Intelligence, Machine Learning, Natural Language Processing, Python	The paper discusses AI techniques for requirement analysis in software engineering, highlighting the use of NLP and ML to transform user requirements into structured specifications for software development.
<b>S3</b>	Model-Based Engineering (MBE), Development Operations (DevOps), Artificial Intelligence (AI), Machine Learning (ML)	The paper presents the AIDORt project, which focuses on AI-augmented automation supporting various stages of Cyber-Physical Systems development, including modeling, coding, testing, and monitoring. It combines MBE principles with AI-enhanced methods and tools to engineer more reliable CPSs.
<b>S4</b>	Machine Learning Algorithms, Prototyping, Validation Techniques	The study proposes a technique for automating the validation of software requirements using machine-learning algorithms to analyze textual data from software requirements specifications. It aims to enhance the requirements validation process by reducing manual effort and improving accuracy.
<b>S5</b>	AI, Usability Evaluation, HCI, OpenCog AI Framework	This paper conceptualizes an AI-augmented usability evaluation framework (AIUEF) aiming to replace end-users with personas for evaluating HCI requirements in SRS, proposing a blueprint for automated usability evaluation using the OpenCog AI framework to improve system design usability.
<b>S6</b>	Machine Learning (ML), Support Vector Machine (SVM), Naïve Bayes, Decision Tree, Logistic Regression, Support Vector Classification (SVC)	The article introduces an ensemble ML technique for classifying functional requirements (FR) to improve accuracy and availability. This technique combines various ML models with enhanced accuracy as a weight in a weighted ensemble voting approach, achieving 99.45% accuracy in classifying FR.

<b>S7</b>	Machine Learning, Rule-Based Classification	The study develops a method for assessing and improving software requirements quality using machine-learning techniques. It focuses on emulating expert judgment to classify requirements and suggest improvements, employing rule-based classifiers and various ML algorithms for a flexible, context-adapted quality assessment.
<b>S8</b>	Deep Learning, Multi-Layer Perceptron (MLP), Convolutional Neural Network (CNN)	This study proposes a cloud-based service utilizing deep learning techniques, specifically MLP and CNN models, for the automated classification of software quality requirements in requirement specifications, aiming to enhance the precision and efficiency of the software development process.
<b>S9</b>	Natural Language Processing (NLP), Deep Learning, Transfer Learning, Convolutional Neural Networks (CNN)	The study presents an approach to automatically detect privacy-related information within user stories using NLP and deep learning techniques, specifically through a transfer learning approach with a pre-trained CNN. This aims to enhance privacy awareness in agile software development by improving the detection of privacy requirements.
<b>S10</b>	Language Transformers, Zero-Shot Classification, Sentiment Analysis	The study presents an algorithm that automates the generation of product requirements from E-Shop consumer reviews using language-transformer technologies. It combines NLP techniques with transformer concepts, including keyword and summary generation, to identify and prioritize critical user needs, showcasing a practical application to the Amazon Echo Dot.
<b>S11</b>	Natural Language Processing (NLP), Semantic Role Labeling, Dependency and Part-of-Speech-Based Approach	The paper presents NLP methodologies for the semi-automated formalization of software requirements, discussing dependency and part-of-speech-based approaches alongside semantic role labeling to generate human- and machine-readable requirement models demonstrated through industrial use cases in the automotive and railway domains.
<b>S12</b>	IoT, AI, Knowledge-enabled AI, Data-driven AI	This article explores the integration of IoT and AI, detailing the advancements and challenges of combining these technologies across various layers of IoT architecture. It highlights the roles of knowledge-enabled and data-driven AI in enhancing IoT capabilities, with a case study on smart cities illustrating practical applications.

<b>S13</b>	UML, Domain-Specific Modeling Languages, jUCMNav, Sirius Framework	The study systematically reviews existing methodologies, models, tools, and techniques used for specifying and modeling requirements in AI systems. It highlights the challenges and limitations of current practices and emphasizes the need for new approaches to support requirements engineering for AI, particularly highlighting the use of UML and domain-specific modeling languages alongside tools like jUCMNav and Sirius for practical applications.
<b>S14</b>	Domain-Specific Language (DSL), Model-Driven Engineering (MDE), Neural Networks, Dataset Engineering	The article presents SEMKIS-DSL, a textual DSL designed to support software engineers in specifying requirements and recognition skills for neural networks. This DSL is part of the SEMKIS methodology, aimed at improving neural network recognition skills through precise specifications of dataset requirements and expected recognition skills, using model-driven engineering principles for automated data generation based on specified requirements.
<b>S15</b>	Automated Planning, Machine Learning, AMLCP Algorithm	The study proposes a novel approach for automating the acquisition of software requirements using AI. It employs automated planning techniques and ML methods to transform requirements into an incomplete planning domain, further refined by the AMLCP algorithm to learn action models with uncertain effects, ultimately converting these into formal software requirement specifications.
<b>S16</b>	NLP Techniques, AI Engine (DizReq), Model-Driven Engineering	The article introduces DizReq, an AI engine that transforms diverse text-based requirement specifications into machine-processable models. It emphasizes automating the generation of context-sensitive user stories, using NLP techniques to address the challenge of understanding and managing requirements across various formats.
<b>S17</b>	AI Ethics, Transparency, Explainability, Ethical Guidelines Analysis, Empirical Study Methods	The paper investigates the transformation of ethical guidelines into practical explainability requirements for AI systems. It analyzes ethical guidelines from 16 organizations to understand the quality requirements emphasized and then evaluates these findings with practitioners to define explainability requirements effectively.
<b>S18</b>	BiLSTM for sequential data processing, CRF for pattern recognition, Knowledge Graphs for data structuring and visualization.	The study specifies a novel approach for automatically generating SRS graphs from textual descriptions, aiming to improve the precision and accuracy of software requirements by using AI technologies.

# C

## **Appendix C. Prompts used to train the GenAI4RSL**

## Appendix C

### Appendix C. 1. User Management System

#### 1.1. Set of Prompts used in translation from original NL to RSL specification

The User Management System was the first attempt to translate NL requirements into RSL specifications. Therefore, there were used prompts for each set of RSL elements. In the end, it was used a prompt for the complete and detailed RSL specification.

- **Basic System Definition**

```
Translate the following system description into a RSL specification for a System definition at the Business Level.  
[The user management system must allow users to create, edit, and delete their profiles, and should send a confirmation email after each action.]
```

**Desired Output Format:**

```
System <System Name> "<System Description>" : Business [  
  description "<System Summary>"  
  ...]
```

- **Stakeholders and Actors**

```
Define Stakeholders and Actors for the system in RSL specification, based on the following NL description.  
[The user management system must allow users to create, edit, and delete their profiles, and should send a confirmation email after each action.]
```

**Desired Output Format:**

```
Stakeholder <Stakeholder ID> "<Name>" : <Type> [  
  description "<Stakeholder Description>"  
Actor <Actor ID> "<Name>" : <Type> [  
  stakeholder <Stakeholder ID>  
  description "<Actor Description>"
```

- **Data Entities**

```
Based on the NL description, define DataEntities and DataAttributes to represent key business objects in RSL specifications.  
[The user management system must allow users to create, edit, and delete their profiles, and should send a confirmation email after each action.]
```

**Desired Output Format:**

```
DataEntity <Entity Name> "<Entity Description>" : <Type> [  
  attribute <Attribute Name> "<Description>" : <DataType> [constraints (...)]  
  ...]
```

- **Use Cases and Actions**

Based on the NL description, define Use Cases and Actions to capture the system's functionality in RSL. Each use case will describe actions that users perform in the system, and data entities involved.  
[The user management system must allow users to create, edit, and delete their profiles, and should send a confirmation email after each action.]

### Desired Output Format:

```
UseCase <UseCase ID> "<UseCase Description>" : <UseCaseType> [  
  primaryActor <Actor ID>  
  dataEntity <DataEntity ID>  
  actions <Action ID>  
  description "<Detailed description of the use case>"]
```

- **Data Entity Clusters**

Based on the NL description define DataEntityClusters in RSL to logically group related data entities that interact with one another in the system.  
[The user management system must allow users to create, edit, and delete their profiles, and should send a confirmation email after each action.]

### Desired Output Format:

```
DataEntityCluster <Cluster ID> "<Cluster Name>" :<Cluster Type> [  
  main <Main DataEntity ID>  
  children <Child DataEntity ID>  
  description "<Detailed description of the cluster>"  
]
```

- **Prompt for complete translation to RSL specification**

Translate the following Natural Language (NL) description into a complete and detailed Requirements Specification Language (RSL) specification. The translation should contain the following elements: Basic System Definition, Stakeholders and Actors, Data Entities, Data Entity Clusters, Use Cases, and Actions.  
[The user management system must allow users to create, edit, and delete their profiles, and should send a confirmation email after each action.]

## 1.2. Set of Prompts used in translation from RSL specification to NL requirements

The User Management System was the first translation from RSL specifications to NL requirements. The prompt used had the purpose to the GenAI4RSL produce in NL a definition of all the elements included in the RSL specifications and should consider the ISO/IEC/IEEE 29148 standards in describing functional and non-functional requirements.

Given the set of requirements in RSL language presented below, define these requirements in natural language, ensuring the specification of the following types of elements: Actors, DataEntities, UseCases, Actions, and StateMachine. This conversion should consider ISO/IEC/IEEE 29148 standards and describe the functional requirements in the use case style and the non-functional requirements (if identified).  
[RSL generated in the next appendix D1 for User Management System]

## Appendix C. 2. Billing System

### 2.1 Set of Prompts used in translation from original NL to RSL specification

The Billing System was the second translation from NL to RSL specifications. The prompt used was similar to the one constructed in the previous system to produce a complete and detailed specification in RSL.

#### Prompt:

Translate the following Natural Language (NL) description into a complete and detailed Requirements Specification Language (RSL) specification. The translation should contain the following elements: Basic System Definition, Stakeholders and Actors, Data Entities, Data Entity Clusters, Use Cases and Actions.  
[BillingSystem **is** a system that allows users to manage customers, products **and** invoices.  
A user of the system **is** someone that has a user account **and is** assigned to user roles, such **as** operator, manager **and** administrator.  
  
User-administrator shall be responsible **for** managing users, configuring technical features (e.g., user roles, export configuration parameters, general enterprise information). System shall allow user-administrator to register users. During this process the administrator shall specify first **and** last name of the user, email address, **and** username. User password shall be automatically generated by the system **and** sent to its user email.  
  
User-operator **is** responsible **for** managing customers **and** invoices. User-operator shall create/update information related to customers **and** invoices. For each customer the system shall maintain the following information: name, fiscal id, logo image, address, bank information **and** additional information such **as** basic person contact information. System shall allow user-operator to define some customers **as** VIP. If a customer **is** defined **as** VIP, **for** each of her invoice, the system shall allow user-operator to set a predefined discount tax. That amount of discount can change throughout the time **and** depends on the current policy of the company. For each product the system shall maintain the following information: name, description, price, VAT category, **and** VAT value. Product must have only one VAT category **and** maintain the respective current VAT value.  
  
User-operator shall create new invoices (**with** respective invoice details). However, before sending an invoice to a customer, the invoice shall be formally approved by the user manager. Only after such approval, the user-operator shall issue **and** send that invoice electronically by e-mail **and** by regular post. In addition, **for** each invoice, the user-operator needs to keep track **if** it **is** paid **or not**.  
  
The System shall automatically alert the user-manager, **for** all the invoices that were sent to customers but **not** yet paid, after 30 days of their respective issue date.  
At the beginning of each year the System shall archive **and** send to the ERP-System all paid invoices of the last year.]

## 2.2 Set of Prompts used in translation from RSL specification to NL requirements

The Billing System was the second translation from RSL specifications to NL requirements. The prompt used had the purpose to the GenAI4RSL produce in NL a definition of all the elements included in the RSL specifications and should consider the ISO/IEC/IEEE 29148 standards in describing functional and non-functional requirements.

### Prompt:

Given the set of requirements in RSL language presented below, define these requirements in natural language, ensuring the specification of the following types of elements: Actors, DataEntities, UseCases, Actions, and StateMachine. This conversion should consider ISO/IEC/IEEE 29148 standards and describe the functional requirements in the use case style and the non-functional requirements (if identified).  
[RSL generated in the next appendix D2 for Billing System]

## Appendix C. 3. LibrarySystem

### 3.1 Set of Prompts used in translation from original NL to RSL specification

The Library System was the third translation from NL to RSL specifications. The prompt was similar to the others previously used, in order to obtain from GenAI4RSL a complete and detailed specification in RSL.

### Prompt:

Translate the following Natural Language (NL) description into a complete and detailed Requirements Specification Language (RSL) specification. The translation should contain the following elements: Basic System Definition, Stakeholders and Actors, Data Entities, Data Entity Clusters, Use Cases and Actions.  
[The Library Management System is an application designed to manage library books and borrowers. It enables librarians to add and organize book records efficiently. Manages detailed book information, including ISBN, title, and author. Add new books to the system through a straightforward process. The primary user responsible for managing book records and borrower interactions. This system streamlines the management of library resources, ensuring organized and accessible book data for librarians]

### 3.2 Set of Prompts used in translation from RSL specification to NL requirements

The Lybrary System was the third translation from RSL specifications to NL requirements. The prompt used had the purpose to the GenAI4RSL produce in NL a definition of all the elements included in the RSL specifications and should consider the ISO/IEC/IEEE 29148 standards in describing functional and non-functional requirements.

### Prompt:

Given the set of requirements in RSL language presented below, define these requirements in natural language, ensuring the specification of the following types of elements: Actors, DataEntities, UseCases, Actions, and StateMachine. This conversion should consider ISO/IEC/IEEE 29148 standards and describe the functional requirements in the use case style and the non-functional requirements (if identified).

## **Appendix C. 4. Appointment Scheduler**

### **4.1 Set of Prompts used in translation from original NL to RSL specification**

The Appointment Scheduler System was the fourth translation from NL to RSL specifications. The prompt was similar to the others previously used, in order to obtain from GenAI4RSL a complete and detailed specification in RSL.

#### **Prompt:**

Translate the following Natural Language (NL) description into a complete and detailed Requirements Specification Language (RSL) specification. The translation should contain the following elements: Basic System Definition, Stakeholders and Actors, Data Entities, Data Entity Clusters, Use Cases and Actions.  
[The Appointment Scheduler System is a platform designed to streamline scheduling, managing, and tracking appointments for a service provider. It supports customers, users, managers, and system administrators in managing appointment-related tasks. The Users can book, cancel, and update appointments. The Managers can add and manage available services. Users and managers can view a list of scheduled appointments. The User Books and manages their appointments. The Manager Adds new services and oversees appointment scheduling. The Customer Receives appointment details for services. The System Administrator Manages technical configurations and user roles.]

### **4.2 Set of Prompts used in translation from RSL specification to NL requirements**

The Appointment Scheduler System was the fourth translation from RSL specifications to NL requirements. The prompt used had the purpose to the GenAI4RSL produce in NL a definition of all the elements included in the RSL specifications and should consider the ISO/IEC/IEEE 29148 standards in describing functional and non-functional requirements.

#### **Prompt:**

Given the set of requirements in RSL language presented below, define these requirements in natural language, ensuring the specification of the following types of elements: Actors, DataEntities, UseCases, Actions, and StateMachine. This conversion should consider ISO/IEC/IEEE 29148 standards and describe the functional requirements in the use case style and the non-functional requirements (if identified).  
[RSL generated in the next appendix D4 for Appointment Scheduler System]

## **Appendix C. 5. Management Accounting System**

### **5.1 Set of Prompts used in translation from original NL to RSL specification**

The Management Accounting System was the fifth translation from NL to RSL specifications. The purpose was to test the GenAI4RSL in a real-world project and the outcome for both types of translations. The prompt was similar to the others previously used, in order to obtain from GenAI4RSL a complete and detailed specification in RSL.

**Prompt:**

Given the set of requirements in natural language (NL) presented below, define these requirements in RSL language, ensuring the specification of the following types of elements: Stakeholders, Actors, DataEntities, UseCases, Actions, and StateMachine.  
[Insert the content of the document "NL1\_CG\_Original Specification\_Requirements\_Revenue\_Items" here]

**5.2 Set of Prompts used in translation from RSL specification to NL requirements**

The Management Accounting System was the fifth translation from RSL specifications to NL requirements. The prompt used had the purpose to the GenAI4RSL produce in NL a definition of all the elements included in the RSL specifications and should consider the ISO/IEC/IEEE 29148 standards in describing functional and non-functional requirements.

**Prompt:**

Given the set of requirements in RSL language presented below, define these requirements in natural language, ensuring the specification of the following types of elements: Actors, DataEntities, UseCases, Actions, and StateMachine. This conversion should consider ISO/IEC/IEEE 29148 standards and describe the functional requirements in the use case style and the non-functional requirements (if identified).  
[RSL generated in the next appendix D5 for Management Accounting System]

# D

## **Appendix D. Datasets used to train the GenAI4RSL**

## Appendix D

### Appendix D. 1. User Management System

#### 1.1 Original NL requirement

The user management system must allow users to create, edit, and delete their profiles, and should send a confirmation email after each action.

#### 1.2 Original RSL Specification

```
System UserManagement "User Profile Management": Application [
description "System to manage user profiles including creation, editing,
deletion, and email notifications." ]

Actor a_User "User" : User [
description "End user managing their profile."
]

DataEntity e_UserProfile "User Profile" : Master [
attribute ID : Integer [constraints (PrimaryKey)]
attribute email : Email [constraints (NotNull Unique)]
attribute Name : String [constraints (NotNull)]
attribute password : String [constraints (NotNull)]
description "DataEntity to represent user profiles."
]

ActionType a_CreateProfile "Create Profile" [
description "Allows creation of a new user profile." ]

ActionType a_EditProfile "Edit Profile" [
description "Allows editing an existing user profile." ]

ActionType a_DeleteProfile "Delete Profile" [
description "Allows deletion of a user profile." ]

ActionType a_SendConfirmationEmail "Send Confirmation Email" [
description "Send a confirmation email after profile actions."
]

UseCase uc_ManageUserProfile "Manage User Profile" : EntitiesManage [
primaryActor a_User
dataEntity e_UserProfile
actions a_CreateProfile, a_EditProfile, a_DeleteProfile,
a_SendConfirmationEmail
description "Allows users to create, edit, or delete their profiles,
and sends a confirmation email after each action." ]
```

### 1.3 Generated RSL Specification

From the prompt described in the previous appendix the GenAI4RSL produced the following RSL specification:

```
Package UserManagementSystem

/*****
  System Definition
  *****/
System UserManagementSystem "User Management System": Application [
  description "A system that allows users to create, edit, and delete their
  profiles. The system sends a confirmation email after each action to ensure
  updates are acknowledged."
]

/*****
  Stakeholders
  *****/
Stakeholder stk_User "User": Person [
  description "Represents individuals who interact with the system to manage
  their profiles."
]

/*****
  Actors
  *****/
Actor a_User "User": User [
  stakeholder stk_User
  description "Interacts with the system to create, edit, and delete their
  profiles."
]

Actor a_System "System": ExternalSystem [
  description "Handles confirmation email notifications for profile actions."
]

/*****
  Data Entities
  *****/
DataEntity e_UserProfile "User Profile": Master [
  attribute ProfileID : Integer [constraints (PrimaryKey)]
  attribute Name : String [constraints (NotNull)]
  attribute email : Email [constraints (NotNull Unique)]
  attribute Password : String [constraints (NotNull Encrypted)]
  attribute IsActive : Boolean [constraints (NotNull Derived("true"))]
  description "Represents the user profile with essential information such as
  name, email, and password."
]

/*****
  Data Entity Clusters
  *****/
DataEntityCluster ec_UserManagement "User Management Cluster": Master [
  main e_UserProfile
  description "Cluster grouping user profiles to streamline profile management
  operations."
]

/*****
  Actions
  *****/
ActionType a_CreateProfile "Create Profile" [
  description "Allows users to create a new profile."
]
```

```

]

ActionType a_EditProfile "Edit Profile" [
    description "Allows users to edit their existing profiles."
]

ActionType a_DeleteProfile "Delete Profile" [
    description "Allows users to delete their profiles."
]

ActionType a_SendConfirmationEmail "Send Confirmation Email" [
    description "Sends a confirmation email after profile actions are completed."
]
/*****
    Use Cases
*****/
UseCase uc_CreateProfile "Create Profile": EntityCreate [
    primaryActor a_User
    dataEntity e_UserProfile
    actions a_CreateProfile, a_SendConfirmationEmail
    description "Allows users to create a new profile and receive a confirmation
email."
]

UseCase uc_EditProfile "Edit Profile": EntityUpdate [
    primaryActor a_User
    dataEntity e_UserProfile
    actions a_EditProfile, a_SendConfirmationEmail
    description "Allows users to edit their profiles and receive a confirmation
email."
]

UseCase uc_DeleteProfile "Delete Profile": EntityDelete [
    primaryActor a_User
    dataEntity e_UserProfile
    actions a_DeleteProfile, a_SendConfirmationEmail
    description "Allows users to delete their profiles and receive a confirmation
email."
]
/*****
    Non-Functional Requirements
*****/
QR r_Security "Security Requirement": Security [
    description "All profile actions must be secured using encryption, and user
authentication must be required."
]

QR r_NotificationReliability "Notification Reliability Requirement": Reliability
[
    description "Confirmation emails must be sent reliably after each profile
action to ensure acknowledgment."
]

QR r_Performance "Performance Requirement": Performance [
    description "Profile actions (create, edit, delete) and confirmation email
notifications must complete within 2 seconds." ]

```

## 1.4 Generated NL Specification

From the prompt described in the previous appendix the GenAI4RSL produced the following RSL specification:

### System Overview

The **User Management System** is an application designed to allow users to manage their profiles. Users can create, edit, and delete their profiles, and the system will send a confirmation email after each action to ensure updates are acknowledged. The system prioritizes security, performance, and notification reliability to provide a robust user experience.

---

### Actors

1. **User:**
    - Represents individuals who interact with the system.
    - Responsibilities:
      - Create, edit, and delete their profiles.
      - Receive confirmation emails for profile actions.
  2. **System:**
    - An external system responsible for handling confirmation email notifications.
- 

### Data Entities

1. **User Profile:**
    - Represents a user's profile containing the following attributes:
      - **ProfileID:** Unique identifier for the profile (Primary Key).
      - **Name:** User's full name (Mandatory).
      - **Email:** User's email address (Mandatory, Unique).
      - **Password:** User's encrypted password (Mandatory).
      - **IsActive:** Boolean indicating if the profile is active. Defaults to true.
  2. **User Management Cluster:**
    - Groups all user profiles to organize and streamline profile management operations.
- 

### Use Cases

The system provides the following core functionalities:

1. **Create Profile:**
  - **Primary Actor:** User
  - **Goal:** Allow users to create a new profile in the system.

- **Precondition:** The user must provide all mandatory information (Name, Email, Password).
  - **Main Actions:**
    1. User submits profile details to the system.
    2. System validates the information and creates the profile.
    3. System sends a confirmation email to the user.
  - **Postcondition:** A new profile is created, and the user receives a confirmation email.
2. **Edit Profile:**
- **Primary Actor:** User
  - **Goal:** Allow users to update their existing profile information.
  - **Precondition:** The user must have an active profile.
  - **Main Actions:**
    1. User submits updated profile details.
    2. System validates the changes and updates the profile.
    3. System sends a confirmation email to the user.
  - **Postcondition:** The user profile is updated, and the user receives a confirmation email.
3. **Delete Profile:**
- **Primary Actor:** User
  - **Goal:** Allow users to delete their profiles from the system.
  - **Precondition:** The user must confirm their intent to delete the profile.
  - **Main Actions:**
    1. User requests to delete the profile.
    2. System verifies the request and deletes the profile.
    3. System sends a confirmation email to the user.
  - **Postcondition:** The user profile is removed, and the user receives a confirmation email.

---

## Actions

1. **Create Profile:**
  - Enables the creation of a new user profile.
2. **Edit Profile:**
  - Allows updates to existing user profile information.
3. **Delete Profile:**
  - Facilitates the removal of a user profile from the system.
4. **Send Confirmation Email:**
  - Ensures users receive a notification confirming the action taken (create, edit, delete).

---

## State Machine

The **User Profile State Machine** describes the lifecycle of a user profile:

1. **Active** (Initial State):
  - Represents an active and operational profile.
  - Transition to **Inactive** occurs when:
    - A user deletes their profile.
  - Transition to **Updated** occurs when:
    - A user edits their profile.
2. **Updated**:
  - Represents a profile after an update.
  - Transition to **Active** occurs upon saving changes.
3. **Inactive**:
  - Represents a deleted profile.

---

## Non-Functional Requirements

1. **Security Requirement**:
  - All profile actions (create, edit, delete) must be secured with encryption.
  - User authentication must be required to access and manage profiles.
2. **Notification Reliability Requirement**:
  - Confirmation emails must be sent reliably after each profile action to ensure the user is informed of the changes.
3. **Performance Requirement**:
  - All profile actions (create, edit, delete) and confirmation email notifications must complete within 2 seconds.

## Appendix D. 2. Billing System [55]

### 2.1 Original NL Specification

BillingSystem **is** a system that allows users to manage customers, products **and** invoices.

A user of the system **is** someone that has a user account **and is** assigned to user roles, such **as** operator, manager **and** administrator.

User-administrator shall be responsible **for** managing users, configuring technical features (e.g., user roles, export configuration parameters, general enterprise information). System shall allow user-administrator to register users. During this process the administrator shall specify **first and** last name of the user, email address, **and** username. User password shall be automatically generated by the system **and** sent to its user email.

User-operator **is** responsible **for** managing customers **and** invoices. User-operator shall create/update information related to customers **and** invoices. For each customer the system shall maintain the following information: name, fiscal id, logo image, address, bank information **and** additional information such **as** basic person contact information. System shall allow user-operator to define some customers **as** VIP. If a customer **is** defined **as** VIP, **for** each of her invoice, the system shall allow user-operator to set a predefined discount tax. That amount of discount can change throughout the time **and** depends on the current policy of the company. For each product the system shall maintain the following information: name, description, price, VAT category, **and** VAT value. Product must have only one VAT category **and** maintain the respective current VAT value.

User-operator shall create new invoices (**with** respective invoice details). However, before sending an invoice to a customer, the invoice shall be formally approved by the user-manager. Only after such approval, the user-operator shall issue **and** send that invoice electronically by e-mail **and** by regular post. In addition, **for** each invoice, the user-operator needs to keep track **if** it **is** paid **or not**.

The System shall automatically alert the user-manager, **for** all the invoices that were sent to customers but **not** yet paid, after 30 days of their respective issue date. In the beginning of each year the System shall archive **and** send to the ERP-System all paid invoices of the last year.

### 2.2 Original RSL Specification

Package BillingSystem

```
/*  
System definition  
*/
```

```
System Billing_App_v6 "BillingSystem (Application Level)" : Application [isFinal  
description
```

```
"  
BillingSystem is a system that allows users to manage customers, products and  
invoices.
```

A user of the system **is** someone that has a user account **and is** assigned to user roles, such **as** operator, manager **and** administrator.

User-administrator shall be responsible **for** managing users, configuring technical features (e.g., user roles, export configuration parameters, general enterprise information). System shall allow user-administrator to register users. During this process the administrator shall specify first **and** last name of the user, email address, **and** username. User password shall be automatically generated by the system **and** sent to its user email.

User-operator **is** responsible **for** managing customers **and** invoices. User-operator shall create/update information related to customers **and** invoices. For each customer the system shall maintain the following information: name, fiscal id, logo image, address, bank information **and** additional information such **as** basic person contact information. System shall allow user-operator to define some customers **as** VIP. If a customer **is** defined **as** VIP, **for** each of her invoice, the system shall allow user-operator to set a predefined discount tax. That amount of discount can change throughout the time **and** depends on the current policy of the company. For each product the system shall maintain the following information: name, description, price, VAT category, **and** VAT value. Product must have only one VAT category **and** maintain the respective current VAT value.

User-operator shall create new invoices (**with** respective invoice details). However, before sending an invoice to a customer, the invoice shall be formally approved by the user-manager. Only after such approval, the user-operator shall issue **and** send that invoice electronically by e-mail **and** by regular post. In addition, **for** each invoice, the user-operator needs to keep track **if** it **is** paid **or not**.

The System shall automatically alert the user-manager, **for** all the invoices that were sent to customers but **not** yet paid, after 30 days of their respective issue date. In the beginning of each year the System shall archive **and** send to the ERP-System all paid invoices of the last year. [...]  
"]

```
/******  
Event view  
*****/  
Event ev1_1stDayOfMonth "1st Day of the Month" : Timer [isCatch]  
Event ev2_InvoicesNotPaid "Alert when invoices not yet paid" : Conditional  
[isCatch]  
Event ev3_OldInvoices "Old Invoices to Archive" : Conditional [isCatch]  
Event ev4_BeginningOfTheYear "Beginning of each Year" : Timer [isCatch]  
  
/******  
DataEntities view  
*****/  
  
DataEnumeration SizeKind values (Small, REG "Regular", Large, ExtraLarge)  
DataEnumeration UserRoleKind values (Admin, Manager, Operator, Customer)  
DataEnumeration InvoiceStatusKind values (PEND "Pending", Approved, REJ  
"Rejected", Issued, Paid, Deleted)  
DataEnumeration VATRateKind values (Standard, Reduced, Special)  
  
DataEntity e_VAT "VAT Category" : Reference [  
attribute Code "Code" : Integer [constraints (PrimaryKey)]  
attribute Rate "Rate" : DataEnumeration VATRateKind [constraints (NotNull)]  
attribute Name "Name" : String(30)
```

```

attribute Value "Value" : Decimal(2.1) [constraints (NotNull)]
description "VAT Categories"]

DataEntity e_Product "Product" : Master [
  attribute ID : Integer [constraints (PrimaryKey)]
  attribute Name : String(50) [constraints (multiplicity "1..2")]
  attribute VATCode : Integer [constraints (NotNull ForeignKey (e_VAT))]
  attribute VATValue : Decimal [constraints (NotNull Derived ("e_VAT.VATValue"))]
]
  attribute Value "Price Without VAT" : Decimal(16.2) [constraints (NotNull) ]
  attribute ValueWithVAT "Price With VAT" : Decimal(16.2) [constraints (NotNull
Derived ("Value * (1+VATValue)") ) ]
  attribute size : DataEnumeration SizeKind
  description "Products"]

DataEntity e_Customer "Customer" : Master [
  attribute ID "Customer ID" : Integer [constraints (PrimaryKey)]
  attribute Name "Name" : String(50) [constraints (NotNull)]
  attribute fiscalID "Fiscal ID" : String(12) [helpMessage "Customer's Fiscal Id"
constraints (NotNull Unique) ]
  attribute BankID "Bank ID" : Regex [helpMessage "IBAN of customer's bank
account" constraints (Check (ck_BankID "ValidBankID(BankID)"))]
  attribute email "Email" : Email
  attribute phone "Phone #" : String(12) [constraints (NotNull)]
  attribute image "Image" : Image
  constraints ( Encrypted Check (ck_Customer1 "ValidFiscalID(fiscalID)") )
  description "Customers"]

DataEntity e_CustomerVIP "CustomerVIP" : Master [
  isA e_Customer
  attribute discountRate "Discount Rate" : Decimal [defaultValue "20%" ]
  constraints (Encrypted)
  description "Customers VIP"]

DataEntity e_Invoice "Invoice" : Document: Regular [
  attribute ID "Invoice ID" : Integer [constraints (PrimaryKey)]
  attribute customer : Integer [constraints (NotNull ForeignKey (e_Customer
onDelete PROTECT))]
  attribute creationDate "Creation Date" : Date [defaultValue "today" constraints
(NotNull)]
  attribute approvalDate "Approval Date" : Date
  attribute dateIssue "Issue Date" : Date
  attribute paidDate "Payment Date" : Date
  attribute Status : DataEnumeration InvoiceStatusKind
  attribute Value "Total Value Without VAT" : Decimal(16.2)
  [constraints (NotNull Derived ("Sum all InvoiceLine->Value of current
invoice"))]
  attribute ValueWithVAT "Total Value With VAT" : Decimal(16.2)
  [constraints (NotNull Derived ("Sum all InvoiceLine->ValueWithVAT of current
invoice"))] ]

DataEntity e_InvoiceLine "Invoice Line" : Document: Weak [
  attribute ID : Integer [constraints (PrimaryKey)]
  attribute invoiceID : Integer [constraints (NotNull ForeignKey (e_Invoice
onDelete CASCADE))]
  attribute order "InvoiceLine Order" : Integer [constraints (NotNull)]
  attribute productID : Integer [constraints (NotNull ForeignKey (e_Product
onDelete PROTECT))]
  attribute quantity "Number of Itens" : Integer
  attribute productVAT : Decimal [constraints (Derived ("productID->VATCode-
>Value"))]

```

```

    attribute productPrice : Decimal [constraints (Derived ("productID->Value"))]
    attribute Value : Decimal [constraints (Derived ("quantity * productPrice"))]
    attribute ValueWithVAT : Decimal [constraints (Derived ("Value *
(1+productVAT)"))]
    description "InvoiceLines"]

DataEntity e_ClosedInvoice "ClosedInvoice" : Transaction [
    attribute ID "Invoice ID" : Integer [constraints (PrimaryKey)]
    attribute customerID "Customer ID" : Integer [constraints (NotNull ForeignKey
(e_Customer onDelete PROTECT))]
    attribute dateCreation "Creation Date" : Date [defaultValue "today" constraints
(NotNull)]
    attribute dateApproval "Approval Date" : Date
    attribute datePaid "Payment Date" : Date
    attribute dateDeleted "Delete Date" : Date
    attribute totalValueWithoutVAT "Total Value Without VAT" : Decimal(16.2)
[constraints (NotNull)]
    attribute totalValueWithVAT "Total Value With VAT" : Decimal(16.2) [constraints
(NotNull) ]
    description "Closed Invoices, for Backup"]

DataEntity e_User "User" : Master [
    attribute ID "ID" : Integer [constraints (PrimaryKey)]
    attribute login "Login" : Regex [constraints (NotNull Unique)]
    attribute password "Password" : Regex [constraints (NotNull Encrypted)]
    attribute firstName "First Name" : String [constraints (NotNull Encrypted)]
    attribute lastName "Last Name" : String [constraints (NotNull Encrypted)]
    attribute email "Email" : Email [constraints (Unique Encrypted)]
    attribute active "IsActive" : Boolean
    attribute userRole "User Role" : DataEnumeration UserRoleKind
    description "Users"]

/*****
    DataEntityCluster view
    *****/
View view_DataEntityClusters: DataEntityClusterView [ec_Customer, ec_Invoice,
ec_Invoice_Simple, ec_Product, ec_Customer]

DataEntityCluster ec_User "Users" : Master [main e_User]
DataEntityCluster ec_Customer "Customers" : Master [main e_Customer description
"ec_Customer" ]
DataEntityCluster ec_Product "Products" : Master [main e_Product uses e_VAT]
DataEntityCluster ec_Invoice "Invoices" : Document [
    main e_Invoice
    child e_InvoiceLine [uses e_Product, e_VAT]
    uses e_Customer]
DataEntityCluster ec_Invoice_Simple "Invoices (Simple)" : Document [main
e_Invoice uses e_Customer]

/*
 * Requirements: UseCases
 */

// Customer Actions
ActionType aClose [description "Close the current Interaction Space (Form,
Windows, Dialog, etc.)"]
ActionType aCancel [description "Cancel the current Interaction Space (Form,
Windows, Dialog, etc.)"]
ActionType aConfirm [description "Confirm the current Interaction Space (Form,
Windows, Dialog, etc.)"]

```

```

ActionType aSearch [description "Define a set of DataEntity items from a search
query"]
ActionType aFilter [description "Define a set of DataEntity items from a logical
filter query"]
ActionType aPrint [description "Print one or more DataEntity items"]
ActionType aSend [description "Send one or more DataEntity items (via a not
defined process)"]

// Specific Customer Actions
ActionType aSend_Invoice [description "Send Invoice to Customer via email"]
ActionType aExport_Invoices [description "Export a selected set of Invoices to a
previously defined format, e.g. xlsx, json, rtf."]
ActionType aPrint_Invoice [description "Print an Invoice according a previously
defined template"]
ActionType aPrint_Invoices [description "Print a selected list of Invoices
according a previously defined template"]
ActionType aConfirmPayment [description "Confirm the Invoice's payment"]
ActionType aApprove [description "Approve Invoice"]
ActionType aReject [description "Do not Approve Invoice"]
ActionType aPrint_Customer [description "Print Customer Profile"]
ActionType aArchiveAndExportInvoices [description "Archive and Export Paid
Invoices"]

/*****
Stakeholders
*****/

Stakeholder stk_User "User" : Person : Person_User [description "(See Glossary)"]
Stakeholder stk_Operator "Operator" : Person : Person_User [isA stk_User
description "(See Glossary)"]
Stakeholder stk_Manager "Manager" : Person : Person_User [isA stk_User
description "(See Glossary)"]
Stakeholder stk_SysAdmin "SystemAdmin" : Person : Person_User [isA stk_User
description "(See Glossary)"]
Stakeholder stk_Customer "Customer" : Organization : Organization_Customer

/*****
Actors
*****/

Actor a_Admin "TechnicalAdmin" : User [stakeholder stk_SysAdmin description
"Manage Users, VAT, etc."]
Actor a_Manager "Manager" : User [description "Approve Invoices, etc."]
Actor a_Operator "Operator" : User [description "Manage Invoices and Customers"]
Actor a_Customer "Customer" : User [stakeholder stk_Customer description "Receive
Invoices to pay"]
Actor a_ERP "ERP" : ExternalSystem [description "Receive info of paid invoices"]

/*****
UseCases
*****/

UseCase uc_1_ManageInvoices "Manage Invoices" : EntitiesManage [
primaryActor a_Operator
dataEntity ec_Invoice
extensionPoints xp_Create, xp_Update, xp_ConfirmPayment, xp_SendInvoices,
xp_Print ]

UseCase uc_1_1_CreateInvoice "Create Invoice" : EntityCreate [
primaryActor a_Operator

```

```

dataEntity ec_Invoice
actions aConfirm, aCancel
extensionPoints xp_CreateCustomer
extends uc_1_ManageInvoices onExtensionPoint xp_Create ]

UseCase uc_1_2_UpdateInvoice "Update Invoice" : EntityUpdate [
  primaryActor a_Operator
  dataEntity ec_Invoice
  actions aConfirm, aCancel
  extensionPoints xp_CreateCustomer
  extends uc_1_ManageInvoices onExtensionPoint xp_Update
  description "Update Invoice, including Invoice-lines, and optionally create
customer " ]

UseCase uc_1_3_SendInvoices "Send Invoices" : EntitiesInteropSendMessage [
  primaryActor a_Operator
  supportingActors a_Customer
  dataEntity ec_Invoice
  actions aSend, aClose
  includes uc_1_4_PrintInvoice
  extends uc_1_ManageInvoices onExtensionPoint xp_SendInvoices ]

UseCase uc_1_4_PrintInvoice "Print Invoice" : EntityReport [
  primaryActor a_Operator
  dataEntity ec_Invoice
  precondition "Invoice.state in {'Approved', 'Issued', 'Paid'}"
  actions aPrint, aClose
  extends uc_1_ManageInvoices onExtensionPoint xp_Print
  description "Print Invoice/Receipt"]

UseCase uc_1_5_RegisterPayment "Register Payment" : EntityUpdate [
  primaryActor a_Operator
  dataEntity ec_Invoice
  precondition "Invoice.state in {'Issued'}"
  postcondition "Invoice.state := 'Paid'"
  actions aConfirm, aCancel
  extends uc_1_ManageInvoices onExtensionPoint xp_ConfirmPayment ]

UseCase uc_1_6_CreateCustomer "Create Customer (in the Invoice context)" :
EntityCreate [
  primaryActor a_Operator
  dataEntity ec_Customer
  actions aConfirm, aCancel
  extends uc_1_1_CreateInvoice onExtensionPoint xp_CreateCustomer
  extends uc_1_2_UpdateInvoice onExtensionPoint xp_CreateCustomer
  description "Create Customer (in the scope of Invoice create or update)" ]

UseCase uc_1_7_DeleteInvoice "Delete Invoice" : EntityDelete [
  primaryActor a_Operator
  dataEntity ec_Invoice
  precondition "Invoice.state in {'Pending'}"
  actions aDelete, aClose
  description "Delete Invoice"]

UseCase uc_2_ApproveInvoices "Approve Invoices" : EntitiesBrowse [
  primaryActor a_Manager
  dataEntity ec_Invoice
  actions aClose, aSearch, aFilter
  extensionPoints xp_ApproveInvoice]

UseCase uc_2_1_ApproveInvoice "Consult and Approve Invoice" : EntityUpdate [

```

```

primaryActor a_Manager
dataEntity ec_Invoice
precondition "Invoice.state in {'Pending'}"
postcondition "Invoice.state = 'Approved' or 'Rejected'"
actions aApprove, aReject, aClose
extends uc_2_ApproveInvoices onExtensionPoint xp_ApproveInvoice]

UseCase uc_3_ReceiveAlertOfInvoicesNotYetPaid "Alert Invoices Not Paid" :
EntitiesInteropSendMessage [
  primaryActor a_Manager
  triggeredBy ev2_InvoicesNotPaid
  dataEntity ec_Invoice_Simple
  precondition "Invoice(s).state in {'Pending'}"
  actions aSend ]

UseCase uc_4_ReceiveClosedInvoices "Receive closed invoices":
EntitiesInteropSendMessage [
  primaryActor a_ERP
  triggeredBy ev4_BeginningOfTheYear
  dataEntity ec_Invoice_Simple
  precondition "Invoice(s).state in {'Paid'}" ]

UseCase uc_6_ManageCustomers "Manage Customers" : EntitiesManage [
  primaryActor a_Operator
  dataEntity ec_Customer
  actions aCreate, aRead, aUpdate, aDelete, aSearch, aFilter, aClose
  extensionPoints Print_Customer, Print_Customers]

UseCase uc_6_1_PrintCustomer "Print Customer Report" : EntityReport [
  primaryActor a_Operator
  dataEntity ec_Customer
  actions aPrint, aClose
  extends uc_6_ManageCustomers onExtensionPoint Print_Customer]

UseCase uc_6_2_PrintCustomers "Print Customers Report" : EntitiesReport [
  primaryActor a_Operator
  dataEntity ec_Customer
  actions aPrint, aClose
  extends uc_6_ManageCustomers onExtensionPoint Print_Customers]

```

## 2.3 Generated RSL Specification

The following RSL specification for the BillingSystem was produced from the original NL requirement.

```

Package BillingSystem

/*****
  System Definition
  *****/
System BillingSystem "Billing System": Application [
  description "A system that allows users to manage customers, products, and
invoices."
]
/*****
  Stakeholders
  *****/
Stakeholder stk_Admin "Administrator": Person [

```

```

        description "Responsible for managing users and configuring technical
features."
    ]

Stakeholder stk_Operator "Operator": Person [
    description "Responsible for managing customers and invoices."
]

Stakeholder stk_Manager "Manager": Person [
    description "Responsible for approving invoices before they are sent to
customers."
]
/*****
    Actors
*****/
Actor a_Admin "Administrator": User [
    stakeholder stk_Admin
    description "Manages users, technical configurations, and enterprise
information."
]

Actor a_Operator "Operator": User [
    stakeholder stk_Operator
    description "Manages customer and invoice information."
]

Actor a_Manager "Manager": User [
    stakeholder stk_Manager
    description "Approves invoices before issuance."
]
/*****
    Data Entities
*****/
DataEntity e_Customer "Customer": Master [
    attribute ID : Integer [constraints (PrimaryKey)]
    attribute Name : String [constraints (NotNull)]
    attribute FiscalID : String [constraints (NotNull Unique)]
    attribute LogoImage : Image
    attribute Address : String
    attribute BankInfo : String
    attribute ContactInfo : String
    attribute IsVIP : Boolean [constraints (NotNull DefaultValue(false))]
    description "Entity representing customers of the company."
]

DataEntity e_Product "Product": Master [
    attribute ID : Integer [constraints (PrimaryKey)]
    attribute Name : String [constraints (NotNull)]
    attribute Description : String
    attribute Price : Decimal(16.2) [constraints (NotNull)]
    attribute VATCategory : String [constraints (NotNull)]
    attribute VATValue : Decimal [constraints (NotNull)]
    description "Entity representing products offered by the company."
]

DataEntity e_Invoice "Invoice": Document [
    attribute ID : Integer [constraints (PrimaryKey)]
    attribute CustomerID : Integer [constraints (NotNull ForeignKey(e_Customer))]
    attribute TotalWithoutVAT : Decimal(16.2) [constraints (NotNull)]
    attribute TotalWithVAT : Decimal(16.2) [constraints (NotNull)]
    attribute Status : String [constraints (DefaultValue("Pending"))]

```

```

        attribute IssueDate : Date [constraints (NotNull DefaultValue("today"))]
        attribute ApprovalDate : Date
        attribute IsPaid : Boolean [constraints (NotNull DefaultValue(false))]
        description "Entity representing invoices issued for customers."
    ]

DataEntity e_InvoiceLine "Invoice Line": Detail [
    attribute ID : Integer [constraints (PrimaryKey)]
    attribute InvoiceID : Integer [constraints (NotNull ForeignKey(e_Invoice))]
    attribute ProductID : Integer [constraints (NotNull ForeignKey(e_Product))]
    attribute Quantity : Integer [constraints (NotNull)]
    attribute TotalWithoutVAT : Decimal(16.2) [constraints (NotNull)]
    attribute TotalWithVAT : Decimal(16.2) [constraints (NotNull)]
    description "Detailed items in an invoice."
]
/*****
    Actions
    *****/
ActionType a_RegisterUser "Register User" [
    description "Allows the administrator to register new users."
]

ActionType a_ManageCustomer "Manage Customer" [
    description "Allows the operator to create, update, and manage customer
information."
]

ActionType a_ManageProduct "Manage Product" [
    description "Allows the operator to create and update product information."
]

ActionType a_CreateInvoice "Create Invoice" [
    description "Allows the operator to create new invoices."
]

ActionType a_ApproveInvoice "Approve Invoice" [
    description "Allows the manager to approve pending invoices."
]

ActionType a_TrackPayment "Track Payment" [
    description "Allows the operator to track whether an invoice has been paid."
]

ActionType a_SendInvoice "Send Invoice" [
    description "Allows the operator to issue and send invoices to customers."
]
/*****
    Use Cases
    *****/
UseCase uc_RegisterUser "Register User": EntityCreate [
    primaryActor a_Admin
    dataEntity e_User
    actions a_RegisterUser
    description "Allows the administrator to register new users by specifying
their name, email, and username."
]

UseCase uc_ManageCustomer "Manage Customer": EntitiesManage [
    primaryActor a_Operator
    dataEntity e_Customer
    actions a_ManageCustomer

```

```

        description "Allows the operator to manage customer details, including VIP
status."
    ]

UseCase uc_ManageProduct "Manage Product": EntitiesManage [
    primaryActor a_Operator
    dataEntity e_Product
    actions a_ManageProduct
    description "Allows the operator to manage product information."
]

UseCase uc_CreateInvoice "Create Invoice": EntityCreate [
    primaryActor a_Operator
    dataEntity e_Invoice
    actions a_CreateInvoice
    description "Allows the operator to create a new invoice with details of
products and customers."
]

UseCase uc_ApproveInvoice "Approve Invoice": EntityUpdate [
    primaryActor a_Manager
    dataEntity e_Invoice
    actions a_ApproveInvoice
    precondition "e_Invoice.Status = 'Pending'"
    description "Allows the manager to approve pending invoices before issuance."
]

UseCase uc_SendInvoice "Send Invoice": EntitiesInteropSendMessage [
    primaryActor a_Operator
    dataEntity e_Invoice
    actions a_SendInvoice
    precondition "e_Invoice.Status = 'Approved'"
    description "Allows the operator to send approved invoices to customers via
email and regular post."
]

UseCase uc_TrackPayment "Track Payment": EntityUpdate [
    primaryActor a_Operator
    dataEntity e_Invoice
    actions a_TrackPayment
    description "Allows the operator to track whether an invoice has been paid."
]

/*****
Non-Functional Requirements
*****/
QR r_Performance "Performance Requirement": Performance [
    description "The system shall process and display updates to customer and
invoice records within two seconds."
]

QR r_Security "Security Requirement": Security [
    description "Only authorized roles shall have access to create, update, or
delete records in the system."
]

QR r_Reliability "Reliability Requirement": Reliability [
    description "The system must ensure data integrity for customer, product, and
invoice information."
]

QR r_Automation "Automation Requirement": Automation [

```

```
description "The system shall automatically notify the manager of unpaid
invoices 30 days after their issue date."
]
```

```
QR r_Archiving "Archiving Requirement": Reliability [
description "The system shall archive and send all paid invoices from the
previous year to the ERP system at the beginning of each year."
]
```

## 2.4 Generated NL Specification

The following NL specification for the BillingSystem was produced from the original RSL.

### General Vision

The **Billing System** is a comprehensive application designed to manage customers, products, and invoices. It facilitates workflows for creating, managing, and tracking financial transactions while supporting integration with external systems, such as ERP platforms, for archiving and reporting purposes. The system is scalable, ensuring future adaptability to organizational needs.

---

### Stakeholders

1. **User:** A generic stakeholder representing all system users.
2. **Operator:** Responsible for managing customers, products, and invoices. Operators handle most day-to-day system operations.
3. **Manager:** Oversees invoice approval and receives alerts for unpaid invoices.
4. **System Administrator (SysAdmin):** Configures technical settings, including user accounts and enterprise details.
5. **Customer:** Represents individuals or organizations interacting with the system through invoices and financial transactions.

---

### Actors

1. **Administrator:** Manages users, system configurations, and enterprise-wide settings.
2. **Operator:** Handles customer and product management, invoice creation, and payment tracking.
3. **Manager:** Approves invoices and interacts with system alerts for unpaid transactions.
4. **Customer:** The recipient of invoices and payments.
5. **ERP:** An external system used for archiving invoices and receiving financial transaction data.

---

### Data Entities

1. **Customer:**
  - o Attributes:
    - **Customer ID:** Unique identifier.
    - **Name:** Name of the customer.
    - **Fiscal ID:** Tax identification number.
    - **Bank ID:** Bank information for transactions.
    - **Email:** Email address for communications.
    - **Phone:** Contact number.
    - **Image:** Profile image or logo.
  - o Constraints:
    - Encrypted storage of sensitive fields.
    - Validation checks for fiscal and bank IDs.

- Description: Represents the organization's or individual customer interacting with the system.
- 2. **CustomerVIP:**
  - A specialized subtype of Customer.
  - Additional Attributes:
    - **Discount Rate:** Default discount applied to invoices.
  - Description: Represents high-value customers eligible for predefined benefits.
- 3. **Product:**
  - Attributes:
    - **Product ID:** Unique identifier.
    - **Name:** Name of the product.
    - **Description:** Detailed product information.
    - **Price Without VAT:** Base price.
    - **Price With VAT:** Computed price including VAT.
    - **VAT Category:** Category for tax application.
    - **VAT Value:** Computed tax value.
    - **Size:** Defined product size (e.g., Small, Regular, Large).
  - Description: Represents goods or services offered for invoicing.
- 4. **VAT:**
  - Attributes:
    - **Code:** Unique identifier for VAT.
    - **Rate:** Applicable VAT rate.
    - **Value:** Computed tax value.
  - Description: Captures VAT categories and rates.
- 5. **Invoice:**
  - Attributes:
    - **Invoice ID:** Unique identifier.
    - **Customer:** Associated customer.
    - **Creation Date:** Date of invoice creation.
    - **Approval Date:** Date of approval.
    - **Issue Date:** Date of issuance.
    - **Paid Date:** Date of payment.
    - **Status:** Current status (e.g., Pending, Approved, Issued, Paid).
    - **Total Value Without VAT:** Base invoice amount.
    - **Total Value With VAT:** Total including VAT.
  - Constraints:
    - Derived fields for financial calculations.
  - Description: Central entity for financial transactions.
- 6. **InvoiceLine:**
  - Attributes:
    - **Invoice Line ID:** Unique identifier.
    - **Invoice:** Associated invoice.
    - **Product:** Linked product.
    - **Quantity:** Number of items.
    - **Total Value Without VAT:** Line-level amount without tax.
    - **Total Value With VAT:** Line-level amount with tax.
  - Description: Represents detailed items in an invoice.
- 7. **ClosedInvoice:**
  - Attributes:
    - **Invoice ID:** Unique identifier.
    - **Customer:** Linked customer.
    - **Date Paid:** Finalized payment date.
    - **Date Deleted:** Archive date.

- **Total Value Without VAT:** Final amount without VAT.
  - **Total Value With VAT:** Final amount with VAT.
  - Description: Archived invoices for backup and external reporting.
- 

#### Actions

1. **Create Invoice:** Enables operators to generate a new invoice.
  2. **Approve Invoice:** Allows managers to approve pending invoices.
  3. **Send Invoice:** Facilitates issuing invoices to customers via email or post.
  4. **Track Payment:** Tracks and confirms payments for issued invoices.
  5. **Archive and Export Invoices:** Archives paid invoices and sends data to the ERP system at the start of each year.
  6. **Print Invoice:** Generates invoice reports based on predefined templates.
  7. **Print Customer:** Produces reports for individual customer profiles.
- 

#### Use Cases

1. **Manage Invoices:**
    - Primary Actor: Operator
    - Description: Allows operators to create, update, approve, send, and track invoices.
    - Extension Points:
      - **Create Invoice:** Adds a new invoice.
      - **Update Invoice:** Modifies invoice details.
      - **Register Payment:** Confirms invoice payment.
      - **Print Invoice:** Generates printable versions.
  2. **Approve Invoices:**
    - Primary Actor: Manager
    - Description: Allows managers to review and approve pending invoices.
    - Preconditions: Invoices must have "Pending" status.
  3. **Alert Invoices Not Paid:**
    - Triggered By: System event for invoices overdue by 30 days.
    - Primary Actor: Manager
    - Description: Notifies managers of unpaid invoices.
  4. **Archive Closed Invoices:**
    - Triggered By: System event at the start of each year.
    - Primary Actor: ERP System
    - Description: Archives all paid invoices and exports data to external systems.
- 

#### Non-Functional Requirements

1. **Performance:**
    - Ensure system responses for customer and invoice operations within 2 seconds.
  2. **Security:**
    - Enforce role-based access control for sensitive operations, ensuring only authorized users can modify records.
  3. **Reliability:**
    - Guarantee data integrity for all financial transactions.
  4. **Automation:**
    - Automatically trigger alerts for unpaid invoices and archival of paid invoices.
  5. **Scalability:**
    - Support increasing volumes of customers, products, and invoices while maintaining system performance.
- 

#### Menu Structure and Interface

1. **Create Invoice:** Available for operators to generate new invoices.
2. **Approve Invoices:** Accessible by managers to review pending invoices.
3. **Manage Customers:** Allows operators to add, update, and delete customer records.

4. **Track Payments:** Facilitates payment tracking for issued invoices.
5. **Archive Invoices:** Automates archival processes at the start of each fiscal year.

## Appendix D. 3. LibrarySystem

### 3.1 Original NL Specification

The Library Management System is an application designed to manage library books and borrowers. It enables librarians to add and organize book records efficiently. Manages detailed book information, including ISBN, title, and author. Add new books to the system through a straightforward process. The primary user responsible for managing book records and borrower interactions. This system streamlines the management of library resources, ensuring organized and accessible book data for librarians

### 3.2 Original RSL Specification

Package LibrarySystem

```
System LibrarySystem "Library Management System": Application [
  description "A system that manages books and borrowers in a traditional
  library."]

/*****
  Stakeholders
  *****/
Stakeholder stk_Librarian "Librarian": Person [
  description "Librarian responsible for managing book records and borrowers."]

Stakeholder stk_Borrower "Borrower": Person [
  description "Library user who borrows books."]

/*****
  Actors
  *****/
Actor a_Librarian "Librarian": User [
  stakeholder stk_Librarian
  description "Manages book records and borrowers."]

Actor a_Borrower "Borrower": User [
  stakeholder stk_Borrower
  description "Library user who borrows and returns books."]

/*****
  Data Entities
  *****/
DataEntity e_Book "Book": Master [
  attribute ISBN : String [constraints (PrimaryKey)]
  attribute Title : String [constraints (NotNull)]
  attribute Author : String [constraints (NotNull)]
  attribute Status : DataEnumeration BookStatus [constraints (NotNull)]
  description "Book entity storing information about library books."
]

DataEnumeration BookStatus values (Available, Borrowed, Reserved)

DataEntity e_Borrower "Borrower": Master [
```

```

    attribute BorrowerID : Integer [constraints (PrimaryKey)]
    attribute Name : String [constraints (NotNull)]
    attribute ContactInfo : String
    description "Entity representing individuals who can borrow books."
]

DataEntity e_Loan "Loan": Transaction [
    attribute LoanID : Integer [constraints (PrimaryKey)]
    attribute ISBN : String [constraints (NotNull ForeignKey(e_Book))]
    attribute BorrowerID : Integer [constraints (NotNull ForeignKey(e_Borrower))]
    attribute LoanDate : Date [constraints (NotNull)]
    attribute DueDate : Date [constraints (NotNull)]
    attribute ReturnDate : Date
    description "Record of each book borrowed by a borrower with loan details."
]

/*****
Data Entity Clusters
*****/
DataEntityCluster ec_LibraryManagement: Master [
    main e_Book
    child e_Loan
    description "Cluster representing books, borrowers, and loans in the
library."

/*****
Actions
*****/

ActionType a_AddBookRecord "Add Book Record" [
    description "Action to add a new book to the library system."

ActionType a_CreateLoanRecord "Create Loan Record" [
    description "Creates a new loan record when a book is borrowed."

ActionType a_UpdateBookStatus "Update Book Status" [
    description "Updates the status of a book when borrowed or returned."

ActionType a_RecordReturn "Record Return" [
    description "Records the return of a borrowed book."

ActionType a_SearchBook "Search Book" [
    description "Allows searching for a book by title, author, or ISBN."

ActionType a_AddBorrowerRecord "Add Borrower Record" [
    description "Registers a new borrower in the system."

/*****
Use Cases
*****/

UseCase uc_AddBook "Add Book": EntityCreate [
    primaryActor a_Librarian
    dataEntity e_Book
    actions a_AddBookRecord
    description "Allows the librarian to add a new book to the library system."

UseCase uc_BorrowBook "Borrow Book": EntityUpdate [
    primaryActor a_Borrower

```

```

    dataEntity e_Loan
    precondition "e_Book.Status = 'Available'"
    actions a_CreateLoanRecord, a_UpdateBookStatus
    description "Allows a borrower to check out an available book.]"

UseCase uc_ReturnBook "Return Book": EntityUpdate [
    primaryActor a_Borrower
    dataEntity e_Loan
    precondition "e_Loan.ReturnDate is Null"
    actions a_RecordReturn, a_UpdateBookStatus
    description "Records the return of a book and updates its availability
status.]"

UseCase uc_SearchBook "Search Book": EntitiesManage [
    primaryActor a_Librarian
    dataEntity e_Book
    actions a_SearchBook
    description "Allows the librarian to search for books by title, author, or
ISBN.]"

UseCase uc_AddBorrower "Add Borrower": EntityCreate [
    primaryActor a_Librarian
    dataEntity e_Borrower
    actions a_AddBorrowerRecord
    description "Allows the librarian to add a new borrower to the system.]"

/*****
    State Machine for Book Availability
    *****/

StateMachine sm_BookStatus "Book Availability": Simple [
    dataEntity e_Book
    description "Types of states that a book can have"
    state Available isFinal onEntry "Initial State"
    useCase uc_BorrowBook action a_CreateLoanRecord nextState Borrowed
    state Borrowed isInitial onEntry "Borrowed State"
    useCase uc_ReturnBook action a_RecordReturn nextState Available state Reserved
    isFinal onExit "Final State"]

/*****
    Menu Structure and Interface
    *****/
View menu_LibrarySystem "Library System Menu": UseCaseView [uc_AddBook,
uc_BorrowBook, uc_ReturnBook, uc_SearchBook, uc_AddBorrower]

/*****
    Non-Functional Requirements
    *****/
QR r_Usability "Usability Requirement": Usability [
    description "The system must provide a simple and intuitive interface for
librarians and borrowers to manage library resources."]

QR r_Performance "Performance Requirement": Performance [
    description "Search operations should return results within 2 seconds for
queries on large book inventories."]

QR r_Security "Security Requirement": Security [
    description "Only authorized librarians should have access to add, update,
and delete records."]

```

```
QR r_Reliability "Reliability Requirement": Reliability [
  description "System should ensure data integrity, especially during loan and
return transactions."]
```

### 3.3 Generated RSL Specification

```
Package LibraryManagementSystem
```

```

/*****
  System Definition
*****/
System LibraryManagementSystem "Library Management System": Application [
  description "An application designed to manage library books and borrowers.
It enables librarians to add and organize book records efficiently while
streamlining the management of library resources."
]
/*****
  Stakeholders
*****/
Stakeholder stk_Librarian "Librarian": Person [
  description "The primary user responsible for managing book records and
borrower interactions."
]
/*****
  Actors
*****/
Actor a_Librarian "Librarian": User [
  stakeholder stk_Librarian
  description "Manages book records and ensures the organization and
accessibility of library resources."
]
/*****
  Data Enumerations
*****/
DataEnumeration BookStatus values (Available, Borrowed, Reserved)

/*****
  Data Entities
*****/
DataEntity e_Book "Book": Master [
  attribute ISBN : String [constraints (PrimaryKey)]
  attribute Title : String [constraints (NotNull)]
  attribute Author : String [constraints (NotNull)]
  attribute Status : DataEnumeration BookStatus [constraints (NotNull
DefaultValue("Available"))]
  description "Entity representing detailed book information, including ISBN,
title, author, and status."
]
/*****
  Data Entity Clusters
*****/
DataEntityCluster ec_LibraryManagement: Master [
  main e_Book
  description "Cluster representing the central entity for managing books and
their statuses in the library."
]
/*****
  Actions

```

```

*****/
ActionType a_AddBookRecord "Add Book Record" [
    description "Action to add a new book to the library system."
]

ActionType a_UpdateBookStatus "Update Book Status" [
    description "Action to update the status of a book based on library
transactions."
]

ActionType a_SearchBook "Search Book" [
    description "Action to search for books based on ISBN, title, or author."
]
/*****
    Use Cases
*****/
UseCase uc_AddBook "Add Book": EntityCreate [
    primaryActor a_Librarian
    dataEntity e_Book
    actions a_AddBookRecord
    description "Allows librarians to add new books to the system through a
straightforward process."
]

UseCase uc_SearchBook "Search Book": EntityRetrieve [
    primaryActor a_Librarian
    dataEntity e_Book
    actions a_SearchBook
    description "Allows librarians to search for books by ISBN, title, or
author."
]

UseCase uc_UpdateBookStatus "Update Book Status": EntityUpdate [
    primaryActor a_Librarian
    dataEntity e_Book
    actions a_UpdateBookStatus
    description "Allows librarians to update the status of books to reflect their
availability, borrowing, or reservation."
]
/*****
    State Machines
*****/
StateMachine sm_BookStatus "Book Lifecycle": Simple [
    dataEntity e_Book
    description "Tracks the lifecycle of books in the library, including
available, borrowed, and reserved states."
    state Available isInitial [
        transition useCase uc_UpdateBookStatus action a_UpdateBookStatus
nextState Borrowed
    ]
    state Borrowed [
        transition useCase uc_UpdateBookStatus action a_UpdateBookStatus
nextState Available
        transition useCase uc_UpdateBookStatus action a_UpdateBookStatus
nextState Reserved
    ]
    state Reserved [
        transition useCase uc_UpdateBookStatus action a_UpdateBookStatus
nextState Borrowed
        transition useCase uc_UpdateBookStatus action a_UpdateBookStatus
nextState Available
    ]
]

```

```

]
]
/*****
Non-Functional Requirements
*****/
QR r_Usability "Usability Requirement": Usability [
  description "The system must provide a simple and intuitive interface for
librarians to manage and access book records efficiently."
]

QR r_Performance "Performance Requirement": Performance [
  description "Search and update operations should execute within two seconds,
even for large datasets."
]

QR r_Scalability "Scalability Requirement": Scalability [
  description "The system must accommodate increasing volumes of book records
and transactions without performance degradation."
]

QR r_DataIntegrity "Data Integrity Requirement": Reliability [
  description "The system must ensure that book data remains accurate and
consistent during all operations."
]

QR r_Security "Security Requirement": Security [
  description "Only authorized librarians should have access to add, update,
and search book records."
]

```

### 3.4 Generated NL Specification

The following NL specification for the LibraryManagementSystem was produced from the original RSL.

#### General Vision

The Library Management System is a software application designed to manage books and borrowers in a traditional library setting. It streamlines book-related operations such as adding new records, borrowing, returning, and tracking book availability.

---

#### Stakeholders

- Librarian: A person responsible for managing book records and borrowers.
  - Borrower: A library user who borrows books from the system.
- 

#### Actors

- Librarian: Represents the primary user managing books and borrowers.
  - Borrower: Represents the user borrowing and returning books in the system.
- 

#### Data Entities

- Book: Stores information about books in the library. Attributes include:
  - ISBN: Unique identifier for each book.
  - Title: Title of the book.
  - Author: Author of the book.
  - Status: Current status of the book (Available, Borrowed, or Reserved).

- **Borrower:** Represents individuals eligible to borrow books. Attributes include:
  - **BorrowerID:** Unique identifier for each borrower.
  - **Name:** Name of the borrower.
  - **ContactInfo:** Contact information of the borrower.
- **Loan:** Tracks the borrowing details of each book. Attributes include:
  - **LoanID:** Unique identifier for the loan record.
  - **ISBN:** Reference to the borrowed book.
  - **BorrowerID:** Reference to the borrower.
  - **LoanDate:** Date the book was borrowed.
  - **DueDate:** Due date for returning the book.
  - **ReturnDate:** Date the book was returned.
- **Data Entity Clusters**
  - **Library Management:** Represents the relationship between books and loans, providing an integrated view of library operations.

#### Actions

- **Add Book Record:** Enables the addition of new books to the system.
- **Create Loan Record:** Creates a record for a book being borrowed.
- **Update Book Status:** Updates the book's status based on operations like borrowing or returning.
- **Record Return:** Records the return of a borrowed book.
- **Search Book:** Searches for books based on title, author, or ISBN.
- **Add Borrower Record:** Adds a new borrower to the system.

#### Use Cases

1. **Add Book:** Allows the librarian to add a new book to the system.
2. **Borrow Book:** Allows a borrower to check out an available book.
  - **Precondition:** The book must have a status of "Available."
3. **Return Book:** Records the return of a borrowed book and updates its status.
  - **Precondition:** The loan's return date must be null.
4. **Search Book:** Enables the librarian to search for books by title, author, or ISBN.
5. **Add Borrower:** Allows the librarian to add new borrowers to the system.

#### State Machine for Book Availability

- **Available:** Initial state for books ready to be borrowed.
  - **Transition:** Borrowing a book moves it to the "Borrowed" state.
- **Borrowed:** Indicates a book currently checked out.
  - **Transition:** Returning a book moves it back to the "Available" state.
- **Reserved:** Represents books reserved by borrowers.

#### Menu Structure and Interface

- **Library System Menu:**
  - Add Book
  - Borrow Book
  - Return Book
  - Search Book
  - Add Borrower

#### Non-Functional Requirements

1. **Usability:** The system must provide a simple and intuitive interface for librarians and borrowers.
2. **Performance:** Search operations should return results within 2 seconds, even for large book inventories.
3. **Security:** Only authorized librarians should be able to add, update, or delete records.

4. Reliability: The system must ensure data integrity during all operations, particularly for loan and return transactions.

## Appendix D. 4. AppointmentScheduler

### 4.1 Original NL Specification

The Appointment Scheduler System is a platform designed to streamline scheduling, managing, and tracking appointments for a service provider. It supports customers, users, managers, and system administrators in managing appointment-related tasks. The Users can book, cancel, and update appointments. The Managers can add and manage available services. Users and managers can view a list of scheduled appointments. The User Books and manages their appointments. The Manager Adds new services and oversees appointment scheduling. The Customer Receives appointment details for services. The System Administrator Manages technical configurations and user roles.

### 4.2 Original RSL Specification

```
//System for scheduling and managing appointments for a service provider
Package AppointmentScheduler

/*****
  Business definition
  *****/
System AppointmentScheduler "AppointmentSchedulerSystem defined at business
Level": Business[isFinal]

/*****
  Stakeholders
  *****/
Stakeholder stk_user "User": Person: Person_User
Stakeholder stk_manager "Manager": Person: Person_User
Stakeholder stk_sys_admin "SystemAdmin": Person: Person_User
// Systems
Stakeholder stk_Appointment "AppointmentSchedulerSystemSystem": System
// Others
Stakeholder stk_customer "Customer": Organization: Organization_Customer

/*****
  DataEntities view
  *****/
DataEnumeration TypeService values(BookAppointment, CancelAppointment,
UpdateAppointment)
DataEntity e_Customer "Customer": Master[attribute ID "Customer ID":
Integer[constraints(PrimaryKey)] attribute Name "Name":
String(50)[constraints(NotNull)] attribute FiscalID "Fiscal ID":
String(12)[helpMessage "Customer's Fiscal Id"
constraints(NotNull Unique)] attribute email "Email": Email attribute Phone
"Phone #": String(12)[constraints(NotNull)] constraints(Encrypted
Check(ck_Customer1 "ValidFiscalID(fiscalID)")) description "Customers"]
DataEntity e_Service "Service": Master[attribute ID "Service ID":
Integer[constraints(PrimaryKey)] attribute Name "Service Name":
String(50)[constraints(NotNull)] attribute Type: DataEnumeration TypeService
description "Service"]
DataEntity e_Appointment "Appointment": Master[attribute ID "Appointment":
Integer[constraints(PrimaryKey)] attribute Customer "Customer ID":
Integer[constraints(ForeignKey(e_Customer))]] attribute service "Service ID":
```

```

Integer[constraints(ForeignKey(e_Service))] attribute date "Appointment date":
Date[defaultValue "today"
constraints(NotNull)] description "Appointment"]

/*****
Actors
*****/
Actor a_Admin "TechnicalAdmin": User[stakeholder stk_sys_admin description
"Manage Users, VAT, etc."]
Actor a_manager "Manager": User[stakeholder stk_manager description "Delete
Appointment, etc."]
Actor a_user "User": User[stakeholder stk_user description "Book and list
Appointments"]
Actor a_customer "Customer": User[stakeholder stk_customer description "Receive
Appointments"]

// Customer Actions
ActionType aCreateAppointment[description "Allows a user to book a new
appointment."]
ActionType aCancelAppointment[description "Allows a user to cancel an existing
appointment."]
ActionType aUpdateAppointment[description "Allows a user to update the details of
an existing appointment."]
ActionType aAddService[description "Allows a manager to add a new service to the
system."]
ActionType aListAppointments[description "Allows users and managers to view a
list of scheduled appointments."]

/*****
Use Cases
*****/
// Use Case: Book Appointment
UseCase uc_1_ManageAppointments "Manage Appointments":
EntitiesManage[primaryActor a_manager dataEntity e_Appointment extensionPoints
xp_CreateAppointment, xp_CancelAppointment, xp_UpdateAppointment, xp_AddService,
xp_ListAppointment]
UseCase uc_BookAppointment "Book Appointment": EntityCreate[primaryActor a_user
dataEntity e_Appointment actions aCreateAppointment description "Allows a user to
book a new appointment."]
// Use Case: Cancel Appointment
UseCase uc_CancelAppointment "Cancel Appointment": EntityDelete[primaryActor
a_user dataEntity e_Appointment actions aCancelAppointment description "c."]
// Use Case: Update Appointment
UseCase uc_UpdateAppointment "Update Appointment": EntityUpdate[primaryActor
a_user dataEntity e_Appointment actions aUpdateAppointment description "Allows a
user to update the details of an existing appointment."]
// Use Case: Add Service
UseCase uc_AddService "Add Service": EntityCreate[primaryActor a_manager
dataEntity e_Service actions aAddService description "Allows a manager to add a
new service to the system."]
// Use Case: List of Appointments
UseCase uc_ListAppointments "List of Appointments": EntityRead[primaryActor
a_user dataEntity e_Appointment actions aListAppointments description "Allows
users and managers to view a list of scheduled appointments."]

```

### 4.3 Generated RSL Specification

Package AppointmentSchedulerSystem

```

/*****
System Definition
*****/
System AppointmentSchedulerSystem "Appointment Scheduler System": Application [
    description "A platform designed to streamline scheduling, managing, and
tracking appointments for a service provider. It supports customers, users,
managers, and system administrators in managing appointment-related tasks."
]
/*****
Stakeholders
*****/
Stakeholder stk_User "User": Person [
    description "Represents individuals who book, cancel, and manage their
appointments."
]

Stakeholder stk_Manager "Manager": Person [
    description "Oversees appointment scheduling and manages available services."
]

Stakeholder stk_Customer "Customer": Organization [
    description "Receives appointment details for the services they interact
with."
]

Stakeholder stk_SysAdmin "System Administrator": Person [
    description "Manages technical configurations and user roles."
]
/*****
Actors
*****/
Actor a_User "User": User [
    stakeholder stk_User
    description "Books and manages their appointments."
]

Actor a_Manager "Manager": User [
    stakeholder stk_Manager
    description "Adds and manages available services and oversees scheduling."
]

Actor a_Customer "Customer": User [
    stakeholder stk_Customer
    description "Receives appointment details for services."
]

Actor a_SysAdmin "System Administrator": User [
    stakeholder stk_SysAdmin
    description "Manages technical configurations and user roles."
]
/*****
Data Entities
*****/
DataEntity e_Appointment "Appointment": Master [
    attribute AppointmentID : Integer [constraints (PrimaryKey)]
    attribute UserID : Integer [constraints (NotNull ForeignKey(e_User))]
    attribute ServiceID : Integer [constraints (NotNull ForeignKey(e_Service))]
    attribute AppointmentDate : DateTime [constraints (NotNull)]
    attribute Status : DataEnumeration AppointmentStatus [constraints (NotNull
DefaultValue("Scheduled"))]

```

```

        description "Tracks details of appointments, including associated users and
services."
    ]

DataEnumeration AppointmentStatus values (Scheduled, Cancelled, Completed)

DataEntity e_Service "Service": Master [
    attribute ServiceID : Integer [constraints (PrimaryKey)]
    attribute ServiceName : String [constraints (NotNull)]
    attribute Description : String
    description "Represents the services offered by the service provider."
]

DataEntity e_User "User": Master [
    attribute UserID : Integer [constraints (PrimaryKey)]
    attribute Name : String [constraints (NotNull)]
    attribute Email : Email
    attribute Role : DataEnumeration UserRole [constraints (NotNull)]
    description "Represents individuals using the system, including users,
managers, and system administrators."
]

DataEnumeration UserRole values (User, Manager, SystemAdministrator)

/*****
    Actions
*****/
ActionType a_BookAppointment "Book Appointment" [
    description "Allows a user to book a new appointment."
]

ActionType a_CancelAppointment "Cancel Appointment" [
    description "Allows a user to cancel an existing appointment."
]

ActionType a_UpdateAppointment "Update Appointment" [
    description "Allows a user to modify the details of an existing appointment."
]

ActionType a_ViewAppointments "View Appointments" [
    description "Allows users and managers to view a list of scheduled
appointments."
]

ActionType a_AddService "Add Service" [
    description "Allows a manager to add new services to the system."
]

ActionType a_ManageConfigurations "Manage Configurations" [
    description "Allows the system administrator to manage technical settings and
user roles."
]
/*****
    Use Cases
*****/
UseCase uc_BookAppointment "Book Appointment": EntityCreate [
    primaryActor a_User
    dataEntity e_Appointment
    actions a_BookAppointment
    description "Enables users to book new appointments."
]

```

```

UseCase uc_CancelAppointment "Cancel Appointment": EntityUpdate [
    primaryActor a_User
    dataEntity e_Appointment
    actions a_CancelAppointment
    description "Enables users to cancel existing appointments."
]

UseCase uc_UpdateAppointment "Update Appointment": EntityUpdate [
    primaryActor a_User
    dataEntity e_Appointment
    actions a_UpdateAppointment
    description "Enables users to update details of their appointments."
]

UseCase uc_ViewAppointments "View Appointments": EntitiesManage [
    primaryActor a_User
    secondaryActor a_Manager
    dataEntity e_Appointment
    actions a_ViewAppointments
    description "Allows users and managers to view a list of scheduled
appointments."
]

UseCase uc_AddService "Add Service": EntityCreate [
    primaryActor a_Manager
    dataEntity e_Service
    actions a_AddService
    description "Allows managers to add new services to the system."
]

UseCase uc_ManageConfigurations "Manage Configurations": EntitiesManage [
    primaryActor a_SysAdmin
    dataEntity e_User
    actions a_ManageConfigurations
    description "Allows system administrators to manage technical settings and
user roles."
]
/*****
    Non-Functional Requirements
*****/
QR r_Usability "Usability Requirement": Usability [
    description "The system must provide a simple and intuitive interface for
users, managers, and administrators."
]

QR r_Performance "Performance Requirement": Performance [
    description "Appointment booking, updating, and viewing must execute within
two seconds."
]

QR r_Security "Security Requirement": Security [
    description "Role-based access control must ensure only authorized users can
access sensitive operations."
]

QR r_Reliability "Reliability Requirement": Reliability [
    description "The system must ensure data consistency during appointment
scheduling and updates."
]

```

## 4.4 Generated NL Specification

The following NL specification for the AppointmentSchedulerSystem was produced from the original RSL.

### General Vision

The Appointment Scheduler System is a business-level system designed to streamline the scheduling and management of appointments for a service provider. It facilitates the organization of services and appointments while supporting user interactions and administrative oversight.

---

### Stakeholders

- **User:** A person who interacts with the system to manage and schedule appointments.
  - **Manager:** A person responsible for overseeing appointments and managing available services.
  - **System Administrator:** A technical administrator who manages user accounts and system configurations.
  - **Customer:** An organization or individual who uses the services and participates in appointments.
- 

### Actors

- **Technical Administrator:** Represents the system administrator responsible for managing users, system configurations, and technical settings.
  - **Manager:** Represents the primary actor managing appointments and services.
  - **User:** Represents individuals booking, canceling, and updating their appointments.
  - **Customer:** Represents the recipient of services, such as receiving appointments.
- 

### Data Entities

- **Customer:** Represents individuals or organizations interacting with the service provider. Attributes:
  - **ID (Customer ID):** A unique identifier for each customer.
  - **Name:** The name of the customer.
  - **Fiscal ID:** A unique tax identifier.
  - **Email:** The email address of the customer.
  - **Phone:** The phone number of the customer.
  - **Constraints:** Data is encrypted and validated to ensure integrity.

- **Service:** Represents the services offered by the provider. Attributes:
  - **ID (Service ID):** A unique identifier for each service.
  - **Name:** The name of the service.
  - **Type:** Specifies the type of service (e.g., Book, Cancel, Update).
  - **Description:** Provides details about the service.
  
- **Appointment:** Tracks the details of appointments scheduled by users.
 

Attributes:

  - **ID (Appointment ID):** A unique identifier for each appointment.
  - **Customer (Customer ID):** References the customer associated with the appointment.
  - **Service (Service ID):** References the service linked to the appointment.
  - **Date:** The date of the appointment.
  - **Constraints:** Appointment dates default to "today" unless specified otherwise.

---

#### Actions

- **Create Appointment:** Allows a user to book a new appointment.
- **Cancel Appointment:** Allows a user to cancel an existing appointment.
- **Update Appointment:** Enables users to modify the details of an existing appointment.
- **Add Service:** Allows a manager to add a new service to the system.
- **List Appointments:** Provides functionality for users and managers to view a list of scheduled appointments.

---

#### Use Cases

- **Manage Appointments:** Overseen by managers, this encompasses creating, canceling, updating, and viewing appointments.
- **Book Appointment:** Enables users to book a new appointment.
  - **Primary Actor:** User
  - **Data Entity:** Appointment
  - **Actions:** Create Appointment
  - **Description:** Allows a user to book a new appointment.
- **Cancel Appointment:** Allows users to cancel an existing appointment.
  - **Primary Actor:** User
  - **Data Entity:** Appointment
  - **Actions:** Cancel Appointment
  - **Description:** Allows a user to cancel an existing appointment.
- **Update Appointment:** Allows users to modify an existing appointment.

- Primary Actor: User
    - Data Entity: Appointment
    - Actions: Update Appointment
    - Description: Allows a user to update the details of an existing appointment.
  - Add Service: Enables managers to add new services to the system.
    - Primary Actor: Manager
    - Data Entity: Service
    - Actions: Add Service
    - Description: Allows a manager to add a new service to the system.
  - List Appointments: Displays all scheduled appointments for users and managers.
    - Primary Actor: User
    - Data Entity: Appointment
    - Actions: List Appointments
    - Description: Provides a list of all scheduled appointments.
- 

#### Non-Functional Requirements

- Usability: The system must provide a user-friendly interface, ensuring ease of navigation for all users.
  - Performance: The system should process appointment-related queries, including searches and updates, within two seconds.
  - Security: Only authorized personnel, such as managers and administrators, should access or modify sensitive data, including customer and appointment records.
  - Reliability: The system must ensure data integrity, especially during appointment scheduling and updates.
- 

#### Menu Structure and Interface

- The system's interface provides the following options:
  - Book Appointment: Enables users to create new appointments.
  - Cancel Appointment: Allows users to cancel existing appointments.
  - Update Appointment: Lets users modify appointment details.
  - Add Service: Facilitates the addition of new services by managers.
  - List Appointments: Displays a list of all appointments for users and managers.

## Appendix D. 5. Management Accounting System

### 5.1 Original NL Specification (NL1\_CG\_Original Specification\_Requirements\_Revenue\_Items)

This set of requirements aims to describe the "Revenue Items" module to be developed within the Management Accounting system, enabling the automated accounting movement of revenues and their reflection in a report. Below are succinct descriptions of each functionality needed to achieve this goal:

- **Requirement 1:** "View Revenue Items": Action to open the window for viewing Revenue Items.
- **Requirement 1:** "Edit Modal": Action to edit Revenue Items through a modal.
- **Requirement 1:** "Validate Events": Validate the existence of events moved with revenue items.
- **Requirement 1:** "Validate Items": Validate if the selected item is elementary in the list of revenue items.
- **Requirement 2:** "Configure Items": Action to associate management accounting elements with revenue items.
- **Requirement 3:** "Import Items": Action to import revenue items from the billing system and financial accounting movements referencing revenue items.
- **Requirement 3:** "Import Movements": Action to import financial accounting movements associated with revenue items.
- **Requirement 4:** "Move Events": Generate automatic movements in management accounting based on the relationship between the financial account and the elements assigned to each revenue item.
- **Requirement 5:** "Unit Types": Action to associate revenue items with unit types and the method for calculating them.
- **Requirement 6:** "Modify Report": Action to include revenue items in the report: **"Proof of accounting revenues entries"**.

---

#### 1. Requirement: Creation of "Revenue Items" Window

**Description:** As a user of the Management Accounting module, I want to view revenue items in a new window.

---

#### Functional Requirement:

- In the menu Entries Configurations, a tab labeled "Revenue" with a sub-tab "Revenue Items" should be created.
  - The Revenue tab should be positioned after the Goods and Services

tab.

- The "Revenue Items" window should include the following fields from the revenue\_items table:
  - Item Code
  - Nature
  - Description
  - Distribution
- For each code with an elementary nature (revenue\_items table and fields nature\_item = "true"), a checkbox should be available before the Code column.
- The window's footer should include an "Edit" option.
  - After selecting the revenue item via the checkbox, the option should be activated.
  - If the revenue items are associated with events posted for the selected management period (reconcilable\_events table and fields: item\_key and posted = "true"), a message should be returned indicating that the revenue item has reconciled events in the selected management period.

---

**Additional Information:**

- The revenue\_items table should be created with the following fields:
  - id
  - item\_key, integer
  - item\_code, string
  - item\_description, string
  - item\_nature, boolean
  - item\_active, boolean
- The reconcilable\_events table should have an additional field: item\_key, integer.

---

**2. Requirement: Configure "Revenue Items"**

**Description:** As a user of the Management Accounting module, I want to configure the revenue items with the Management Accounting structure through a new modal.

---

**Functional Requirement:**

- Accessing the Edit option should open a centered modal with the following fields:
  - Code (field: item\_code)
  - Description (field: item\_description)
  - Below should be two tabs labeled "Revenues" and "Costs"

- For the "Revenues" tab, the following fields should exist:
  - Management Type
  - Revenue Nature
  - Environmental Type
  - Management accounting distribution, which should only be available if the revenue nature is direct.
- For the "Costs" tab, the following fields should exist:
  - Management Type
  - Cost Type
  - Environmental Type
  - Management accounting distribution, based on the selected cost type.
- The modal footer should include two options: Save and Cancel.
  - The Save option saves the distributions in the revenue\_item\_classification and revenue\_item\_distribution tables. For each set of two classifications, they should be saved in two separate ids in the revenue\_item\_classification table, with the consequent distinction of filled fields in the revenue\_item\_distribution table.
  - The Cancel option should close the centered modal without saving the data.

---

**Additional Information:**

- The following tables should be created:
  - The first table should be named revenue\_item\_classification, with the following fields:
    - id
    - revenue\_item\_id, integer
    - management\_type: Enum ManagementType
    - revenue\_nature: Enum RevenueNature
    - environmental\_type: Enum EnvironmentalType
    - cost\_type: Enum CostType
    - costing\_type, integer
    - management\_period\_id, integer
    - item\_nature: Enum ItemNature
    - cost\_classification: Enum CostClassification
  - The second table should be named revenue\_item\_distribution, with the following fields:

- id
  - revenue\_item\_classification\_id, integer
  - organic\_classification\_id, integer
  - organic\_classification\_percentage, numeric(16, 2)
  - activity\_id, integer
  - activity\_percentage, numeric(16, 2)
  - cost\_center\_id, integer
  - cost\_center\_percentage, numeric(16, 2)
- The following tables and respective fields should be associated with the editing of revenue items:
- OrganicClassification Table
    - ID: Integer
    - code: Integer
    - description: String
  - Activity Table
    - id: Integer
    - code: Integer
    - description: String
  - CostCenter Table
    - id: Integer
    - code: Integer
    - description: String
    - cost\_center\_type: Enum CostCenterType
- The following Enums should be considered for the fields of the referenced tables above:
- Enum CostCenterType values (AF "Fixed Asset", PROD "Product", SERV "Service")
  - Enum ItemNature values (ELEMENTAR, COMPOSITE)
  - Enum ManagementType values (COR "Current", AMB "Environmental")
  - Enum EnvironmentalType values (INVEST "Associated with Investments", ACQUIS "Acquisition of raw materials and labor", SER "Social and Environmental Responsibility", CRED "Credits generated with greenhouse gas reduction")
  - Enum RevenueNature values (DIRECT, GENERAL)
  - Enum CostType values (DC "Direct Costs", ICOC "Indirect Costs to organic classifications", ICA "Indirect Costs to activities", ICC "Indirect Costs to cost centers", NC "Non-

incorporable Costs", DCAA "Direct Costs to auxiliary activities")

- Enum CostClassification values (Fixed, Variable)
- Enum RevenueType values (TAXES "Taxes and Fees", SALES "Sales of Goods and Services", SUBSIDIES "Transfers and Subsidies Obtained", OTHER "Others")
- Enum MovementType values (DEB "Debit", CRED "Credit")

---

### 3. Requirement: Importing "Revenue Items"

**Description:** As a user of the Management Accounting module, I want the list of revenue items to be imported through the OLD service, which will request the database for revenue item data from the Billing System, via the synchronization service.

---

#### Functional Requirement:

- In the Configuration menu, in the erp2000 Connector tab, a new option should be added: Revenue Items, following the patrimony items.
- The synchronization of this option should execute the procedure and save the data in the revenue\_items table.
- The method for obtaining the list of revenue items from the billing system should validate if they already exist in the revenue\_items table.
- At the end of the synchronization process, the system should save the status as "imported".

---

### 4. Requirement: Automate Management Accounting Movement Based on "Revenue Items"

**Description:** As a user of the Management Accounting module, I want to automate the movement of revenues in Management Accounting derived from Financial Accounting, through the configurations of revenue items.

---

#### Functional Requirement:

- In the Configuration menu, in the erp2000 Connector tab, when executing the synchronization process for the option: Automatic Entries, the following control should be performed:
  - Based on the value, financial account, and item\_key resulting from the synchronization and saved in the reconcilable\_events table, the correspondence between the financial account plan and the management accounting account plan should be identified in the account\_plan\_correspondence table. The financial account filters the movement to be performed by management type, environmental type, revenue nature, and potential distribution of cost centers existing

in the `revenue_item_classification` and `revenue_item_distribution` tables that correspond to the revenue item, from the `revenue_items` table, for the `item_key` field, and which also appears in the `reconcilable_events` table.

- If the Value field is negative, the movement will be performed for each management accounting account inversely to the definition of the movement type field in the `account_plan_correspondence` table and the inverse field of the movements table should be = 1, during the movement.
- Initially, the `event_state` field in the `reconcilable_events` table should be "false" and at the end of the automatic entry, the field should change to "true".

---

**Additional Information:**

- The following tables should be considered for mapping the movement in management accounting:
  - `Financial_Account_Plan`
    - `financial_account`: String
    - `financial_account_description`: String
    - `management_period`: Date [default "Current Year"]
  - `CG_Account_Plan`
    - `cg_account`: String
    - `cg_account_description`: String
    - `management_period`: Date [default "Current Year"]
  - `Account_Plan_Correspondence`
    - `id`: Integer
    - `financial_account`: String [foreign key (Financial\_Account\_Plan)]
    - `cg_account`: String [foreign key (CG\_Account\_Plan)]
    - `movement_type`: Enum MovementType
    - `inverse`: Boolean
    - `management_type`: Enum ManagementType
    - `cost_type`: Enum CostType
    - `cost_classification`: Enum CostClassification
    - `item_nature`: Enum RevenueNature
    - `revenue_nature`: Enum RevenueNature
    - `revenue_type`: Enum RevenueType
  - `Entry`
    - `id`: Integer [primary key]

- entry\_number: Integer
- reference: String
- document\_type: Enum DocumentType
- Movements
  - id: Integer [primary key]
  - entry\_id: Integer [foreign key (e\_Entry)]
  - movement\_type: Enum MovementType
  - account\_plan: String
  - value: Decimal
- Reconcilable\_Events
  - item\_key: Integer [foreign key (e\_RevenueItems)]
  - event\_state: Enum EventState
  - financial\_account: String [foreign key (Account\_Plan\_Correspondence)]
  - value: Decimal
  - inverse: Boolean
  - entry\_id: Integer [foreign key (Entry)]
- The following Enums should be considered for the fields of the referenced tables above:
  - Enum EventState values (TRUE "Posted", FALSE "Not Posted")
  - Enum DocumentType values (REVENUES, COSTS)

---

## 5. Requirement: "Revenue Items" as Unit Types

**Description:** As a user of the Management Accounting module, I want to associate revenue items with unit types, so they can be considered for the count of units produced or provided for products or services.

---

### Functional Requirement:

- In the Plans and Structures menu, in the Unit Types tab, in the New and Edit options, the following elements should be available:
  - In the Unit Type Calculation dropdown, the option "Revenue Items" should be added.
    - If this option is selected, the Criteria dropdown should provide a multiple-choice list with the revenue items (revenue\_items table).
  - For calculating the values of the selected revenue items, the monthly sum should be performed, which is present in the reconcilable\_events table and has the item\_key referring to each of the selected revenue items.

- The values should be saved in the unit\_type\_calculation table.

---

**Additional Information:**

- The table relationships for associating revenue items and unit types are as follows:
  - Unit\_Type
    - id: Integer [primary key]
    - code: Integer
    - description: String
    - calculation\_type: Enum CalculationTypes
  - Unit\_Type\_Calculation
    - id: Integer [primary key]
    - unit\_type\_id: Integer [foreign key (Unit\_Type)]
    - calculation\_criteria: String
    - description: String
  - The following Enums should be considered for the fields of the referenced tables above:
    - Enum CalculationTypes values (MANUAL "Manual", OS "Service Orders", MAT "Materials", REV "Revenue Items")

---

**6. Requirement: "Proof of Revenue Entries" Report to Incorporate "Revenue Items"**

**Description:** As a user of Management Accounting, I want the revenue items to be included in the proof of revenue entries report, for controlling the revenue moved in the system.

---

**Functional Requirement:**

- In the Reports menu, the map "Proof of Accounts Receivable Entries" should be renamed to "Proof of Integrated Revenue Entries".
- The report should include an additional column, labeled "Item" (following the "Date" column), returning the "code" and "revenue item".
  - In the query, besides the currently returned data, it should also return the data of entries that have associated revenue items in the reconcilable\_events table, in the item\_key field. To obtain the code and description of the revenue items, the item\_key should be related to the key field in the revenue\_items table.
    - The distribution of revenue items in management accounting can be obtained from the revenue\_item\_classification and revenue\_item\_distribution tables.
  - The report's filter area should include a dropdown component, labeled

"Revenue Items", with multiple-choice listing of revenue items, showing the code and description of the revenue item.

- If items are selected, the query should return the entries associated with the selected items.
- In the "Not Posted" option, events to be reconciled with associated revenue items that do not yet have an entry should also be included (reconcilable\_events table, with the item\_key field filled and with the posted field = true).
- The export should also include the new column and respective aggregated data.

---

**7. Requirement: "Results by Sold Products or Provided Services" Report to Incorporate "Revenue Items"**

**Description:** As a user of Management Accounting, I want the revenue items to be reflected in the Results Calculation report, for inclusion as produced or provided units of products or services.

---

**Functional Requirement:**

- In the Reports menu, the map "Results by Sold Products or Provided Services" should include in the column labeled "No. of Units" the count of units associated with products and services, calculated for each unit type, with the definition of revenue items.
  - In the query, for each monthly period, it should read the unit\_type table and the cost\_centers table for the unit\_type\_id field and return the calculated monthly value.

## **5.2 Original RSL Specification (RSL2\_GenAI4RSL\_CG\_Refined Specification\_Revenue Items)**

```
/*
System definition
*/
System ManagementAccountingModule "Management Accounting Module" : Application
[isFinal
description "System that allows the import, visualization, editing of revenue
items, and movement according to the Management Accounting structure."
]
/*
Event view
*/
Event ev1_ImportRevenueItems "Event for importing Revenue Items data" :
Conditional [isCatch]
Event ev2_ImportRevenueItemsMovements "Event for importing movements referencing
Revenue Items" : Conditional [isCatch]
Event ev3_AutomateRevenueItemsMovements "Event for automating movements in
management accounting with reference to the assignment of Revenue Items" :
Conditional [isFinal]
```

```

Event ev4_UnitTypeItems "Event for associating revenue items to types of produced
units" : Conditional [isFinal]
/*****
DataEntities
*****/
DataEntity e_RevenueItems "Revenue Items" : Master [
  attribute ID : Integer [constraints (PrimaryKey)]
  attribute item_key : Integer [constraints (NotNull)]
  attribute item_code : String [constraints (NotNull)]
  attribute item_description : String [constraints (NotNull)]
  attribute item_nature : DataEnumeration ItemNature [constraints (NotNull)]
  attribute item_active : Boolean [constraints (NotNull)]
  description "Table of revenue items with specific fields for accounting
management."
]
DataEntity e_OrganicClassification "Organic Units" : Master [
  attribute ID : Integer [constraints (PrimaryKey)]
  attribute code : Integer [constraints (NotNull)]
  attribute Description : String [constraints (NotNull)]
  description "Table with the structure of the entity's organic units."
]
DataEntity e_Activity "Activities" : Master [
  attribute ID : Integer [constraints (PrimaryKey)]
  attribute code : Integer [constraints (NotNull)]
  attribute Description : String [constraints (NotNull)]
  description "Table with the structure of activities carried out by the
entity."
]
DataEnumeration CostCenterType values (AF "Fixed Asset", PROD "Product", SERV
"Service")

DataEntity e_CostCenter "Cost Centers" : Master [
  attribute ID : Integer [constraints (PrimaryKey)]
  attribute code : Integer [constraints (NotNull)]
  attribute Description : String [constraints (NotNull)]
  attribute cost_center_type : DataEnumeration CostCenterType [constraints
(NotNull)]
  description "Table with the structure of products, services, and fixed assets
produced, provided, and developed by the entity."
]
DataEnumeration ItemNature values (ELEMENTAR,COMPOSTO)
DataEnumeration ManagementType values (COR "Current", AMB "Environmental")
DataEnumeration EnvironmentalType values (INVEST "Associated with Investments",
AQUIS "Acquisition of raw materials and labor", RESP "Social and Environmental
Responsibility", CRED "Credits generated with greenhouse gas reduction")
DataEnumeration RevenueNature values (DIRETO,GERAL)
DataEnumeration CostType values (CD "Direct Costs", CICO "Indirect Costs to
organic classifications", CIA "Indirect Costs to activities", CIC "Indirect Costs
to cost centers", CN "Non-incorporable Costs", CDAA "Direct Costs to auxiliary
activities")
DataEnumeration CostClassification values (Fixed,Variable)
DataEnumeration RevenueType values (IMP "Taxes and Fees", VEND "Sales of Goods
and Services", TRSUB "Transfers and Subsidies Obtained", OUT "Others" )
DataEnumeration MovementType values (DEB "Debit", CRED "Credit")

DataEntity e_RevenueItemsClassification "Revenue Items Classification" : Master [
  attribute ID : Integer [constraints (PrimaryKey)]
  attribute revenue_item_id : Integer [constraints (NotNull ForeignKey
(e_RevenueItems))]
  attribute management_type : DataEnumeration ManagementType
  attribute environmental_type : DataEnumeration EnvironmentalType

```

```

    attribute revenue_nature : DataEnumeration RevenueNature
    attribute cost_type : DataEnumeration CostType
    attribute cost_classification : DataEnumeration CostClassification
    attribute revenue_type : DataEnumeration RevenueType
    attribute management_period : Date [defaultValue "Current Year" constraints
(NotNull)]
    description "Table for classifying revenue items with specific fields
defining the type and nature of revenues for management and automation in
accounting."
]
DataEntity e_RevenueItemsDistribution "Revenue Items Distribution" : Master [
    attribute ID : Integer [constraints (PrimaryKey)]
    attribute revenue_item_classification_id : Integer [constraints (NotNull
ForeignKey (e_RevenueItemsClassification))]
    attribute organic_classification : Integer [constraints (ForeignKey
(e_OrganicClassification))]
    attribute organic_classification_percentage : Decimal
    attribute activity : Integer [constraints (ForeignKey (e_Activity))]
    attribute activity_percentage : Decimal
    attribute cost_center : Integer [constraints (ForeignKey (e_CostCenter))]
    attribute cost_center_percentage : Decimal
    description "Table for distributing revenue items according to the management
accounting structure."
]
DataEntity e_FinancialChartAccounts "Financial Chart of Accounts": Master [
    attribute financial_account : String [constraints (PrimaryKey)]
    attribute financial_account_description : String [constraints (NotNull)]
    attribute management_period : Date [defaultValue "Current Year" constraints
(NotNull)]
    description "Table of the financial accounting chart of accounts."
]
DataEntity e_ManagementAccountingChartAccounts "Management Accounting Chart of
Accounts": Master [
    attribute cg_account : String [constraints (PrimaryKey)]
    attribute cg_account_description : String [constraints (NotNull)]
    attribute management_period : Date [defaultValue "Current Year" constraints
(NotNull)]
    description "Table of the management accounting chart of accounts."
]
DataEntity e_ChartAccountsCorrespondence "Chart of Accounts Correspondence":
Reference [
    attribute ID : Integer [constraints(PrimaryKey)]
    attribute financial_account : String [constraints (NotNull
ForeignKey(e_FinancialChartAccounts))]
    attribute cg_account : String [constraints (NotNull
ForeignKey(e_ManagementAccountingChartAccounts))]
    attribute movement_type : DataEnumeration MovementType
    attribute inverse : Boolean
    attribute management_type : DataEnumeration ManagementType
    attribute cost_type : DataEnumeration CostType
    attribute cost_classification : DataEnumeration CostClassification
    attribute revenue_nature : DataEnumeration RevenueNature
    attribute revenue_type : DataEnumeration RevenueType
    description "Table for the correspondence between financial and management
accounting charts of accounts."
]
DataEnumeration EventState values (TRUE "Posted",FALSE "Not Posted")

DataEntity e_ReconcilableEvents "Reconcilable Events" : Master [
    attribute revenue_item_key : Integer [constraints (ForeignKey
(e_RevenueItems))]

```

```

        attribute event_state : DataEnumeration EventState
        attribute financial_account : String [constraints (ForeignKey
(e_ChartAccountsCorrespondence))]
        attribute Value : Decimal [constraints (NotNull)]
        attribute inverse : Boolean
        attribute entry_id : Integer [constraints (ForeignKey(e_Entry))]
        description "Table of events with an additional field for linking to revenue
items for automatic movement in management accounting."
    ]
DataEntity e_Entry "Entries" : Master [
    attribute ID : Integer [constraints (PrimaryKey)]
    attribute entry_number : Integer [constraints (NotNull)]
    attribute reference : String [constraints(NotNull)]
    attribute document_type : DataEnumeration DocumentType [constraints(NotNull)]
    description "Table identifying entries in management accounting."
]
DataEntity e_Movements "Movements" : Master [
    attribute ID : Integer [constraints (PrimaryKey)]
    attribute entry_id : Integer [constraints (ForeignKey(e_Entry))]
    attribute movement_type : DataEnumeration MovementType [constraints(NotNull)]
    attribute account_plan : String [constraints(NotNull)]
    attribute Value : Decimal [constraints(NotNull)]
    description "Table of movements in management accounting."
]
DataEnumeration DocumentType values (REVENUE,EXPENSES)
DataEnumeration CalculationTypes values (MANUAL "Manual",SO "Service Orders",MAT
"Materials",REV "Revenue Items")

DataEntity e_UnitType "Unit Types" : Master [
    attribute ID : Integer [constraints (PrimaryKey)]
    attribute code : Integer [constraints (NotNull)]
    attribute Description : String [constraints (NotNull)]
    attribute calculation_type : DataEnumeration CalculationTypes
    description "Table of Unit Types for produced quantities and provided
services."
]
DataEntity e_UnitTypeCalculations "Unit Type Calculations" : Reference [
    attribute ID : Integer [constraints (PrimaryKey)]
    attribute unit_type_ID : Integer [constraints (ForeignKey(e_UnitType))]
    attribute calculation_criteria : String [constraints(NotNull)]
    attribute Description : String [constraints(NotNull)]
    description "Table of Unit Type Calculations with the association of material
codes, service orders, and revenue items."
]
/*****
DataEntity Cluster
*****/
DataEntityCluster ec_RevenueItems : Master [
    main e_RevenueItems
    child e_RevenueItemsClassification
    child e_RevenueItemsDistribution
]
DataEntityCluster ec_AutomateMovements : Master [
    main e_ChartAccountsCorrespondence
    child e_RevenueItemsClassification
    child e_RevenueItemsDistribution
]
DataEntityCluster ec_UnitTypes : Master [
    main e_UnitType
    child e_UnitTypeCalculations
]

```

```

DataEntityCluster ec_Movements : Master [
  main e_Entry
  child e_Movements
]
DataEntityCluster ec_RevenueItemsDistribution "Users" : Master [
  main e_RevenueItemsDistribution
  child e_OrganicClassification
  child e_Activity
  child e_CostCenter]
/*****
Stakeholders
*****/
Stakeholder stk_User "User" : Person : Person_User
[description "User of the Management Accounting module interested in viewing
revenue items and automatic movement in management accounting."]
Stakeholder stk_Administrator "Administrator" : Person : Person_User
[description "Administrator with permissions to import, edit revenue items, and
synchronize movement automation."]
/*****
Actors
*****/
Actor a_User "User" : User [stakeholder stk_User description "Views revenue
items."]
Actor a_Administrator "Administrator" : User [stakeholder stk_Administrator
description "Imports, edits revenue items, and triggers the automatic movement
process"]
/*****
UseCases
*****/
UseCase uc_1_ManageRevenueItems "Manage Revenue Items": EntitiesManage [
  primaryActor a_Administrator
  dataEntity e_RevenueItems
  precondition "Authentication required, access restricted to the
Administrator"
  actions a_ImportRevenueItems
  extensionPoints xp_Create, xp_Update
  description "Allows the Administrator to import revenue items from the
billing system and insert them into the Revenue Items table."
]
UseCase uc_1_1_ImportRevenueItems "Import Revenue Items": EntityCreate [
  primaryActor a_Administrator
  triggeredBy ev1_ImportRevenueItems
  dataEntity e_RevenueItems
  precondition "RevenueItems.state in {'Empty'}"
  postcondition "RevenueItems.state in {'Imported'}"
  actions a_ImportRevenueItems
  extends uc_1_ManageRevenueItems onExtensionPoint xp_Create
  description "From the synchronization process in the 'erp2000 Connector' menu,
create a new option: Revenue Items, validate the list of existing items in
management accounting and compare it with the new list obtained. Items not found
in the first list are inserted (DataEntity e_RevenueItems)."
]
UseCase uc_1_2_ImportRevenueItemsMovements "Import movements of Revenue Items":
EntityUpdate [
  primaryActor a_Administrator
  triggeredBy ev2_ImportRevenueItemsMovements
  dataEntity e_ReconcilableEvents
  precondition "ReconcilableEvents.state in {'Empty'}"
  postcondition "ReconcilableEvents.state in {'revenue_item_key'}"
  actions a_ImportRevenueItemsMovements
  extends uc_1_ManageRevenueItems onExtensionPoint xp_Update

```

```

    description "From the synchronization process for event reconciliation,
    validate the list of movements and revenue items from financial accounting and
    compare it with the new list obtained. If the revenue items or movements are not
    in the first list, they are inserted (DataEntity e_ReconcilableEvents)."
  ]
UseCase uc_1_3_AutomateAccountingMovements "Generate movements in management
accounting from the Distribution of Revenue Items": EntityCreate [
  primaryActor a_Administrator
  triggeredBy ev3_AutomateRevenueItemsMovements
  dataEntity ec_AutomateMovements
  precondition "ReconcilableEvents.state in {'To post'}"
  postcondition "ReconcilableEvents.state in {'Posted'}"
  actions aAutomateMovements
  extends uc_1_ManageRevenueItems onExtensionPoint xp_Create
  description "From the synchronization process for movement automation, generate
  entries and movements based on the relationship between financial accounts
  (DataEntity e_ChartOfAccountsCorrespondence) and the assignments of revenue items
  (DataEntityCluster ec_RevenueItems) existing in management accounting."
]
Constraint c_MovementInverses "Moves Inverses" : Other [
  isNegative
  stakeholder stk_Administrator
  priority High
  description "If the reconcilable event returns with a negative value, then
  the movement automation should assume the inverse movement to the profile
  (DataEntity e_ChartOfAccountsCorrespondence)."
]
UseCase uc_2_ViewRevenueItems "View Revenue Items" : EntitiesManage [
  primaryActor a_User
  dataEntity e_RevenueItems
  actions aViewModal,
  aEditItem
  extensionPoints xp_Create, xp_Update
  description "Allows the user to view the revenue items with their respective
  assignments, as well as the items associated with reconciled or to-be-reconciled
  events in the management period. This visualization will be in the modal from the
  'Configurations' menu and the 'Revenue' tab."
]
UseCase uc_2_1_ViewRevenueItems "View Revenue Items" : EntitiesMapShow [
  primaryActor a_User
  dataEntity e_RevenueItems
  actions aViewModal
  description "Allows the user to view the revenue items with their respective
  assignments, as well as the items associated with reconciled or to-be-reconciled
  events in the management period. This visualization will be in the modal from the
  'Configurations' menu and the 'Revenue' tab."
]
UseCase uc_2_2_EditRevenueItems "Edit Revenue Items" : EntityCreate [
  primaryActor a_Administrator
  dataEntity ec_RevenueItems
  actions aEditItem
  extends uc_2_ViewRevenueItems onExtensionPoint xp_Create
  description "Allows the administrator to edit and assign (DataEntityCluster
  ec_RevenueItems) revenue items if they are elementary and not associated with
  reconciled events in the management period. The editing can be done after
  selecting the revenue item and pressing the 'Edit' button in the modal from the
  'Configurations' menu and the 'Revenue' tab."
]
UseCase uc_2_3_EditRevenueItems "Edit Revenue Items" : EntityUpdate [
  primaryActor a_Administrator
  dataEntity ec_RevenueItems

```

```

    actions aEditItem
    extends uc_2_ViewRevenueItems onExtensionPoint xp_Update
    description "Allows the administrator to edit and assign (DataEntityCluster
ec_RevenueItems) revenue items if they are elementary and not associated with
reconciled events in the management period. The editing can be done after
selecting the revenue item and pressing the 'Edit' button in the modal from the
'Configurations' menu and the 'Revenue' tab."
]
UseCase uc_2_4_ValidateRevenueItems "Validate Revenue Items" : EntitiesManage [
    primaryActor a_Administrator
    dataEntity e_ReconcilableEvents
    precondition "Revenue items belong to the list of items (DataEntity
e_RevenueItems)."
```

```

    actions aValidateEvents
    description "Validation process for moved events (DataEntity
e_ReconcilableEvents, attribute posted = 'true' and revenue_item_key is not null)
with associated revenue items in the management period. If there are movements
associated with the revenue item, the editing will not allow changing the revenue
item."
]
Constraint c_RevenueItemsMovement "Items with Movements" : Other [
    isConcrete
    stakeholder stk_User
    priority High
    description "Reconciled events in the posted state containing the revenue
item key, then the revenue item with the same key cannot be changed in the
editing."
]
UseCase uc_2_5_ValidateRevenueItems "Validate Revenue Items" : EntitiesManage [
    primaryActor a_Administrator
    dataEntity e_RevenueItems
    precondition "Revenue items belong to the list of items (DataEntity
e_RevenueItems)."
```

```

    actions aValidateItems
    description "Validation process for selecting the revenue item defined as
elementary (DataEntity RevenueItems, attribute item_nature = 'E'). In the modal
from the 'Configurations' menu and the 'Revenue' tab, only elementary revenue
items will be available for selection."
]
UseCase uc_3_UnitTypes "Unit Types" : EntitiesManage [
    primaryActor a_User
    dataEntity ec_UnitTypes
    actions aUnitTypes
    description "Allows the user to associate revenue items with unit types for
reflection in financial reports for produced or provided units. This association
can be done in the 'Structures and Plans' menu in the 'Unit Types' tab."
]
UseCase uc_4_GenerateRevenueEntryReport "Generate Report with Revenue Entries" :
EntityReport [
    primaryActor a_User
    dataEntity ec_RevenueItemsDistribution
    precondition "The user must be authenticated."
    postcondition "Report successfully generated, including revenue items data."
    actions aGenerateReport
    description "Allows the user to generate a report of revenue entries,
including associated revenue items. The report generation should be in the
'Reports' menu and the report: 'Results Calculation'."
]
UseCase uc_4_1_FilterRevenueItems "Filter Revenue Items" : EntitiesMapShow [
    primaryActor a_User
    dataEntity e_RevenueItems

```

```

        precondition "The user must be authenticated."
        postcondition "Revenue items successfully filtered."
        actions aFilterItems
        description "Allows the user to filter the revenue items in the report.
Filtering will be available in the 'Reports' menu and the report: 'Revenue
Entries'."
    ]
UseCase uc_5_GenerateResultsReport "Generate Report with Results by products and
services" : EntityReport [
    primaryActor a_User
    dataEntity ec_UnitTypes
    precondition "The user must be authenticated."
    postcondition "Report successfully generated, including revenue items data."
    actions aGenerateReport
    description "Allows the user to generate a Results report by products and
services, associated with revenue items. The report generation should be in the
'Reports' menu and the report: 'Results Calculation'."
]
/*****
    Actions
    *****/
ActionType aViewModal "View Modal" [description "Action to open the modal for
viewing Revenue Items."]
ActionType aEditItem "Edit Modal" [description "Action to edit Revenue Items."]
ActionType aValidateEvents "Validate Events" [description "Validates the
existence of moved events with revenue items."]
ActionType aValidateItems "Validate Items" [description "Validates if the
selected item is elementary in the list of revenue items (DataEntity
e_RevenueItems, attribute item_nature)."]
ActionType aImportRevenueItems "Import Items" [description "Action for importing
revenue items from the billing system and movements in financial accounting
with reference to revenue items."]
ActionType aImportRevenueItemsMovements "Import Movements" [description "Action
for importing movements from financial accounting associated with revenue
items."]
ActionType aAutomateMovements "Automate Events" [description "Automatic movement
in management accounting by the relationship between the financial account
and the classification and distribution assigned to each revenue item."]
ActionType aUnitTypes "Unit Types" [description "Action to associate revenue
items with unit types and their calculation methods."]
ActionType aGenerateReport "Generate Report" [description "Include revenue items
data in the management accounting report."]
ActionType aFilterItems "Filter Items" [description "Possibility to filter
revenue items to be included in the management accounting report."]
/*****
    DataEntity's StateMachines
    *****/
StateMachine sm_e_RevenueItems "StateMachine_RevenueItems" : Complex [
    dataEntity e_RevenueItems
    description "StateMachine for the synchronization process with the import of
Revenue Items and associated movements from financial accounting."
    state StateInitial isInitial onEntry "Empty"
        useCase uc_1_1_ImportRevenueItems action aCreate nextState ImportState
        state ImportState isFinal onEntry "e.state = 'Imported'; e.getRevenueItems =
True"
        useCase uc_1_2_ImportRevenueItemsMovements action aCreate nextState
ImportState
        state ImportState isFinal onEntry "e.state = 'revenue_item_key';
e.getRevenueItemsMovements = True"
    ]

```

```

StateMachine sm_e_Movement "StateMachine_RevenueItems" : Complex [
  dataEntity ec_Movements
  description "StateMachine for the synchronization process with the automatic
entry and movement registration in management accounting."
  state StateInitial isInitial onEntry "To post"
    useCase uc_1_3_AutomateAccountingMovements action aCreate nextState
RegisterMovement
  state RegisterMovement isFinal onEntry "e.state = 'Posted'; aAutomateEvents =
true"
]

```

### 5.3 Generated RSL Specification

```
Package RevenueItemsModule
```

```

/*****
  System Definition
*****/
System RevenueItemsModule "Revenue Items Module": Application [
  description "The Revenue Items Module automates accounting movements of
revenues and integrates them into reports. It provides functionalities for
viewing, configuring, importing, and associating revenue items."
]

/*****
  Stakeholders
*****/
Stakeholder stk_ManagementAccountant "Management Accountant": Person [
  description "Represents individuals using the Management Accounting system to
manage revenue items."
]

/*****
  Actors
*****/
Actor a_User "User": User [
  stakeholder stk_ManagementAccountant
  description "Interacts with the system to view, configure, import, and
associate revenue items."
]

Actor a_System "System": ExternalSystem [
  description "Handles synchronization and communication with external systems
like the Billing System."
]

/*****
  Enumerations
*****/
DataEnumeration ItemNature values (ELEMENTARY, COMPOSITE)
DataEnumeration ManagementType values (CURRENT, ENVIRONMENTAL)
DataEnumeration RevenueNature values (DIRECT, GENERAL)
DataEnumeration EventState values (TRUE "Posted", FALSE "Not Posted")
DataEnumeration CalculationTypes values (MANUAL, SERVICE_ORDERS, MATERIALS,
REVENUE_ITEMS)
DataEnumeration CostType values (DEBIT,CREDIT)
DataEnumeration EnvironmentalType values (ENERGIASOLAR,CREDIT)

/*****

```

```

Data Entities
*****/
DataEntity e_RevenueItem "Revenue Item": Master [
  attribute ID : Integer [constraints (PrimaryKey)]
  attribute ItemKey : Integer
  attribute ItemCode : String
  attribute ItemDescription : String
  attribute itemNature : DataEnumeration ItemNature
  attribute ItemActive : Boolean
  description "Represents an individual revenue item with properties such as
code, nature, and activity status."
]

DataEntity e_RevenueItemClassification "Revenue Item Classification": Master [
  attribute ID : Integer [constraints (PrimaryKey)]
  attribute RevenueItemID : Integer [constraints (ForeignKey(e_RevenueItem))]
  attribute managementType : DataEnumeration ManagementType
  attribute revenueNature : DataEnumeration RevenueNature
  attribute environmentalType : DataEnumeration EnvironmentalType
  attribute costType : DataEnumeration CostType
  description "Classifies revenue items into management accounting categories."
]

DataEntity e_RevenueItemDistribution "Revenue Item Distribution": Master [
  attribute ID : Integer [constraints (PrimaryKey)]
  attribute RevenueItemClassificationID : Integer [constraints
(ForeignKey(e_RevenueItemClassification))]
  attribute OrganicClassificationID : Integer
  attribute OrganicClassificationPercentage : Decimal
  attribute ActivityID : Integer
  attribute ActivityPercentage : Decimal
  attribute CostCenterID : Integer
  attribute CostCenterPercentage : Decimal
  description "Details the distribution of revenue items in management
accounting."
]

DataEntity e_ReconcilableEvent "Reconcilable Event": Transaction [
  attribute ItemKey : Integer [constraints (ForeignKey(e_RevenueItem))]
  attribute eventState : DataEnumeration EventState
  attribute FinancialAccount : String
  attribute Value : Decimal
  attribute Inverse : Boolean
  attribute EntryID : Integer [constraints (ForeignKey(e_RevenueItem))]
  description "Represents events associated with revenue items, ready to be
reconciled with financial accounts."
]

/*****
Data Entity Clusters
*****/
DataEntityCluster ec_RevenueItems "Revenue Items Cluster": Master [
  main e_RevenueItem
  child e_RevenueItemClassification
  description "Groups revenue items and their associated classifications,
distributions, and events."
]

/*****
Actions
*****/

```

```

ActionType a_ViewRevenueItems "View Revenue Items" [
    description "Allows users to view revenue items in a dedicated window."
]

ActionType a_EditRevenueItem "Edit Revenue Item" [
    description "Allows users to edit revenue items through a modal."
]

ActionType a_ValidateEvents "Validate Events" [
    description "Validates the existence of events associated with revenue
items."
]

ActionType a_ValidateItems "Validate Items" [
    description "Validates if a revenue item is elementary."
]

ActionType a_ConfigureRevenueItems "Configure Revenue Items" [
    description "Allows users to configure revenue items with management
accounting classifications."
]

ActionType a_ImportRevenueItems "Import Revenue Items" [
    description "Imports revenue items from the Billing System into the
Management Accounting system."
]

ActionType a_MoveEvents "Move Events" [
    description "Automates movements in management accounting based on revenue
items."
]

ActionType a_AssociateUnitTypes "Associate Unit Types" [
    description "Associates revenue items with unit types for calculating
production or service values."
]

ActionType a_ModifyReport "Modify Report" [
    description "Includes revenue items in the 'Proof of Accounting Revenue
Entries' report."
]

/*****
    Use Cases
*****/
UseCase uc_ViewRevenueItems "View Revenue Items": EntitiesMapShow [
    primaryActor a_User
    dataEntity e_RevenueItem
    actions a_ViewRevenueItems
    description "Allows users to view revenue items and their properties in a
dedicated interface."
]

UseCase uc_EditRevenueItem "Edit Revenue Item": EntityUpdate [
    primaryActor a_User
    dataEntity e_RevenueItem
    actions a_EditRevenueItem, a_ValidateEvents, a_ValidateItems
    description "Allows users to edit revenue items through a modal and validate
their properties."
]

```

```

UseCase uc_ConfigureRevenueItems "Configure Revenue Items": EntityCreate [
  primaryActor a_User
  dataEntity e_RevenueItemClassification
  actions a_ConfigureRevenueItems
  description "Allows users to configure revenue items by assigning
classifications and distributions."
]

UseCase uc_ImportRevenueItems "Import Revenue Items": EntitiesInteropImport [
  primaryActor a_System
  dataEntity e_RevenueItem
  actions a_ImportRevenueItems
  description "Imports revenue items from external systems and updates the
local database."
]

UseCase uc_MoveEvents "Move Events": EntitiesMapInteract [
  primaryActor a_System
  dataEntity e_ReconcilableEvent
  actions a_MoveEvents
  description "Generates automated accounting movements based on revenue items
and financial accounts."
]

UseCase uc_AssociateUnitTypes "Associate Unit Types": EntityUpdate [
  primaryActor a_User
  dataEntity e_RevenueItem
  actions a_AssociateUnitTypes
  description "Allows users to associate revenue items with unit types for
production and service calculations."
]

UseCase uc_ModifyReport "Modify Report": EntitiesReport [
  primaryActor a_User
  dataEntity e_RevenueItem
  actions a_ModifyReport
  description "Updates reports to include revenue items in financial entries
and unit calculations."
]

/*****
State Machine
*****/
StateMachine sm_RevenueItemState "Revenue Item State": Complex [
  dataEntity e_RevenueItem
  description "Manages the lifecycle states of revenue items."
  state Active isInitial onEntry "Empty"
    useCase uc_EditRevenueItem action a_EditRevenueItem nextState Configured
  state Configured
    useCase uc_MoveEvents action a_MoveEvents nextState Reconciled
  state Reconciled
    useCase uc_ModifyReport action a_ModifyReport
  state a_ModifyReport isFinal
]

```

## 5.4 Generated NL specification

The following NL specification for the `ManagementAccountingSystem` was produced from the original RSL.

## 1. Overview

The Management Accounting Module is an application that allows for the import, viewing, editing of revenue items, and movement according to the management accounting structure. This system is used to manage revenue items, their classifications, distributions, and automated and integrated movements.

---

## 2. Glossary

- **Revenue Items:** Items representing the entity's revenues.
  - **Movements:** Records of transactions associated with revenue items.
  - **Cost Center:** Control units where costs are accumulated.
  - **Organic Units:** Organizational structure of the entity.
  - **Activities:** Set of tasks carried out by the entity.
  - **ERP2000:** System from which revenue item data is imported.
- 

## 3. Stakeholders

- **User:** Person using the Management Accounting module. Interested in viewing revenue items and automatic movement in management accounting.
  - **Administrator:** Person responsible for importing, editing revenue items, and synchronizing the automation of movements.
- 

## 4. Actors

- **User:** Interacts with the system to view revenue items, define unit types, and obtain reports with revenue item information.
  - **Administrator:** Interacts with the system to import and edit revenue items, as well as to trigger automatic movement processes.
- 

## 5. Data Entities

- **Revenue Items (e\_RevenueItems)**
  - **Type:** Master
  - **Attributes:**
    - ID: Integer (Primary Key)
    - item\_key: Integer (Not Null)
    - item\_code: String (Not Null)
    - item\_description: String (Not Null)
    - item\_nature: DataEnumeration ItemNature (Not Null)
    - item\_active: Boolean (Not Null)
  - **Description:** Table containing detailed information about revenue items for management accounting.
- **Organic Units (e\_OrganicUnits)**

- **Type:** Master
- **Attributes:**
  - ID: Integer (Primary Key)
  - code: Integer (Not Null)
  - description: String (Not Null)
- **Description:** Table structuring the entity's organic units.
- **Activities (e\_Activities)**
  - **Type:** Master
  - **Attributes:**
    - ID: Integer (Primary Key)
    - code: Integer (Not Null)
    - description: String (Not Null)
  - **Description:** Table with the structure of activities carried out by the entity.
- **Cost Centers (e\_CostCenters)**
  - **Type:** Master
  - **Attributes:**
    - ID: Integer (Primary Key)
    - code: Integer (Not Null)
    - description: String (Not Null)
    - cost\_center\_type: DataEnumeration CCType (Not Null)
  - **Description:** Table with the structure of products, services, and fixed assets produced, provided, and developed by the entity.
- **Revenue Item Classification (e\_RevenueItemClassification)**
  - **Type:** Master
  - **Attributes:**
    - ID: Integer (Primary Key)
    - revenue\_item\_id: Integer (Not Null, Foreign Key to e\_RevenueItem)
    - management\_type: DataEnumeration ManagementType
    - environmental\_type: DataEnumeration AmbientType
    - item\_nature: DataEnumeration IncomeType
    - cost\_type: DataEnumeration CostType
    - cost\_classification: DataEnumeration CostClassification
    - revenue\_nature: DataEnumeration RevenueNature
    - management\_period: Date (Default Value: "Current Year", Not Null)
  - **Description:** Table for classifying revenue items with specific

fields for management accounting and automation.

- **Revenue Item Distribution (e\_RevenueItemDistribution)**
  - **Type:** Master
  - **Attributes:**
    - ID: Integer (Primary Key)
    - revenue\_item\_classification\_id: Integer (Not Null, Foreign Key to e\_RevenueItemClassification)
    - organic\_classification: Integer (Foreign Key to e\_OrganicUnits)
    - organic\_classification\_percentage: Decimal
    - activity: Integer (Foreign Key to e\_Activities)
    - activity\_percentage: Decimal
    - cost\_center: Integer (Foreign Key to e\_CostCenter)
    - cost\_center\_percentage: Decimal
  - **Description:** Table for distributing revenue items according to the management accounting structure.
- **Financial Chart of Accounts (e\_FinancialChartAccounts)**
  - **Type:** Master
  - **Attributes:**
    - financial\_account: String (Primary Key)
    - financial\_account\_description: String (Not Null)
    - management\_period: Date (Default Value: "Current Year", Not Null)
  - **Description:** Table of the financial accounting chart of accounts.
- **Management Accounting Chart of Accounts (e\_ManagementAccountingChartAccounts)**
  - **Type:** Master
  - **Attributes:**
    - cg\_account: String (Primary Key)
    - cg\_account\_description: String (Not Null)
    - management\_period: Date (Default Value: "Current Year", Not Null)
  - **Description:** Table of the management accounting chart of accounts.
- **Chart of Accounts Correspondence (e\_ChartAccountsCorrespondence)**
  - **Type:** Master
  - **Attributes:**
    - ID: Integer (Primary Key)
    - financial\_account: String (Foreign Key to

- e\_FinancialChartAccounts)
            - cg\_account: String (Foreign Key to e\_ChartAccountsCorrespondence)
            - movement\_type: DataEnumeration MovementType
            - inverse: Boolean
            - management\_type: DataEnumeration ManagementType
            - cost\_type: DataEnumeration CostType
            - cost\_classification: DataEnumeration CostClassification
            - item\_nature: DataEnumeration RevenueNature
            - revenue\_nature: DataEnumeration RevenueNature
            - revenue\_type: DataEnumeration RevenueType
  - **Description:** Table for the correspondence between financial and management accounting charts of accounts.
- **Entry (e\_Entry)**
  - **Type:** Master
  - **Attributes:**
    - ID: Integer (Primary Key)
    - entry\_number: Integer (Not Null)
    - reference: String (Not Null)
    - document\_type: DataEnumeration DocumentType (Not Null)
  - **Description:** Table identifying entries in management accounting.
- **Movement (e\_Movement)**
  - **Type:** Master
  - **Attributes:**
    - ID: Integer (Primary Key)
    - entry\_id: Integer (Foreign Key to e\_Entry)
    - movement\_type: DataEnumeration MovementType (Not Null)
    - account\_plan: String (Not Null)
    - value: Decimal (Not Null)
  - **Description:** Table identifying movements in management accounting.
- **Reconcilable Events (e\_ReconcilableEvents)**
  - **Type:** Master
  - **Attributes:**
    - ID: Integer (Primary Key)
    - item\_key: Integer (Foreign Key to e\_RevenueItems)
    - event\_state: DataEnumeration EventState (Not Null)
    - financial\_account: String (Foreign Key to e\_ChartAccountsCorrespondence)

- value: Decimal (Not Null)
- inverse: Boolean
- entry\_id: Integer (Foreign Key to e\_Entry)
- **Description:** Table identifying events from financial accounting for posting in management accounting.

---

## 6. Data Entity Clusters

DataEntity Clusters are logical groupings of various data entities that represent a cohesive set of interrelated data within a system.

- **Revenue Items (ec\_RevenueItems)**
  - Main: e\_RevenueItems
  - Details: e\_RevenueItemsClassification
  - Details: e\_RevenueItemsDistribution
- **Automate Movements (ec\_AutomateMovements)**
  - Main: e\_ChartAccountsCorrespondence
  - Details: e\_RevenueItemsClassification
  - Details: e\_RevenueItemsDistribution
- **Unit Types (c\_UnitTypes)**
  - Main: e\_UnitTypes
  - Details: e\_UnitTypesCalculation
- **Movements (ec\_Movements)**
  - Main: e\_Entry
  - Details: e\_Movements

---

## 7. Functional Requirements

### 7.1 View Revenue Items

**Description:** Allows the user to view the list of revenue items and their assignments in Management Accounting.

- **Main Actor:** User
- **Data Entities Used:** RevenueItems
- **Actions:** View Modal, Edit Item
- **Where and How:** This functionality is accessed in the "Configurations" menu, "Revenue" tab, where the user can open the viewing window.

### 7.2 Edit Revenue Items

**Description:** Allows the administrator to edit and assign Organic Classifications, Activities, and Cost Centers of Management Accounting to revenue items, provided they are not associated with reconciled events in the management period.

- **Main Actor:** Administrator

- **Data Entities Used:** RevenueItems, Organic Classification, Activities, Cost Centers, Revenue Item Classification, Revenue Item Distribution.
- **Actions:** View Modal, Edit Item, Modify Item
- **Restrictions:** Items with movements cannot be modified.
- **Where and How:** Available in the "Configurations" menu, "Revenue" tab. After selecting a revenue item, the administrator can edit the details through the edit modal.

### 7.3 Import Revenue Items

**Description:** Allows the administrator to import revenue items from the billing system and insert them into the Revenue Items table.

- **Main Actor:** Administrator
- **Data Entities Used:** e\_RevenueItems
- **Pre-condition:** Authentication required, access restricted to the administrator.
- **Post-condition:** Revenue items imported and updated in the Revenue Items table.
- **Actions:** Import Items
- **Where and How:** The import action occurs in the "ERP2000 Connector" menu with the option "Import Revenue Items". After selection, the obtained list of items is compared with the existing one, and if not present in the latter, new items are added to management accounting.

### 7.4 Import Revenue Item Movements

**Description:** Validates the list of existing movements and revenue items in Management Accounting and compares it with the new list obtained. If the movements are not in the first list, they are inserted.

- **Main Actor:** Administrator
- **Data Entities Used:** e\_ReconcilableEvents
- **Pre-condition:** ReconcilableEvents.state in {'Empty'}
- **Post-condition:** ReconcilableEvents.state in {'item\_key'}
- **Actions:** Import Movements
- **Where and How:** This functionality is accessed through the "ERP2000 Connector" menu in the "Reconcile Events" option to import financial accounting movements containing Revenue Items and not present in the system.

### 7.5 Automate Movements in Management Accounting

**Description:** Automates movements in management accounting based on the relationship between financial accounts and revenue item assignments.

- **Main Actor:** Administrator

- **Data Entities Used:** ec\_AutomateMovements
- **Pre-condition:** ReconcilableEvents.state in {'To post'}
- **Post-condition:** ReconcilableEvents.state in ('Posted')
- **Actions:** Move Events
- **Constraints:** Events with negative values will be subject to reverse movement.
- **Where and How:** This action is initiated in the "Configurations" menu and in the "ERP2000" tab, in the "Automatic Entries" option for automating movements after importing financial accounting movements with associated revenue items. The entries and respective movements are generated based on the related data between the revenue items, financial accounts, and Management Accounting configurations.

### 7.6 Unit Types

**Description:** Allows the user to associate revenue items with calculation units to reflect them in a financial report for produced or provided units. This association can be made in the Plans and Structures menu, in the "Unit Types" tab.

- **Main Actor:** User
- **Data Entities Used:** UnitTypes, Revenue Items
- **Actions:** Associate Items with Unit Types
- **Where and How:** Available in the "Plans and Structures" menu, "Unit Types" tab. The creation of unit type calculations associating revenue items.

### 7.7 Generate Revenue Entry Report

**Description:** Allows the user to generate a revenue entry report, including associated revenue items. The report generation should be available in the 'Reports' menu and in the report: "Results Calculation".

- **Main Actor:** User
- **Data Entities Used:** Revenue Items, Reconcilable Events, Entries, Movements.
- **Actions:** Generate Report
- **Where and How:** Available in the "Reports" menu, in the "Proof of accounting revenues entries" report. The creation of the report containing the revenue items data.

### 7.8 Filter Revenue Items

**Description:** Allows the user to filter revenue items in the report. Filtering will be available in the 'Reports' menu and in the report: "Revenue Entries".

- **Main Actor:** User
- **Data Entities Used:** RevenueItems

- **Actions:** Filter Revenue Items
- **Where and How:** In the "Proof of accounting revenues entries" report, there should be a dropdown component with the list of revenue items, allowing the user to select and filter the revenue items data in the report.

### 7.9 Generate Results Report

**Description:** Allows the user to generate a Results report by products and services associated with revenue items. The report generation should be available in the "Reports" menu and in the report: "Results Calculation".

- **Main Actor:** User
- **Data Entities Used:** RevenueItems
- **Actions:** Filter Revenue Items
- **Where and How:** In the "Results by Sold Products or Services Provided in the Period" report, unit types will be included for each product and service, with the possibility of associating revenue items that reflect the dynamics of produced or provided units.

---

## 8. Non-Functional Requirements

### 8.1 Usability

**Description:** The system should be intuitive and easy to use, allowing users to efficiently view and edit revenue items.

- **Criterion:** The interface should be clear, with well-defined menus and intuitive navigation options.

### 8.2 Security

**Description:** Ensure that only authenticated administrators can import and edit revenue items.

- **Criterion:** Implementation of robust authentication and role-based access control.

### 8.4 Performance

**Description:** The system should be able to efficiently process large volumes of data imports.

- **Criterion:** Import time should not exceed 5 minutes for up to 10,000 records.

### 8.5 Reliability

**Description:** The system should ensure the integrity of imported and processed data.

- **Criterion:** Implementation of validation and data verification mechanisms during import and movement processes.

---

## 9. Menu and Tabs Structure

- **Configurations Menu and "ERP2000 Connector" Tab:** Data synchronization functionality:
  - **Option "Revenue Items":** Used to import new revenue items.
  - **Option "Reconcile Events":** Used to import financial movements associated with revenue items.
  - **Option "Automatic Entries":** Used to automate revenue movements in management accounting based on rules defined in revenue items.
- **Configurations Menu:**
  - **Revenue Tab:** Allows viewing, editing, and classifying revenue items, as well as seeing associated assignments.
- **Plans and Structures Menu:**
  - **Unit Types Tab:** Allows assigning revenue items to unit types to be considered in reports.
- **Reports Menu:**
  - **"Proof of Accounts Receivable Entries" Report:** Provides information on movements performed in management accounting related to revenue items.
  - **"Results by Sold Products or Services Provided in the Period" Report:** Provides information on the quantities of sold products or provided services based on revenue items.

# E

## **Appendix E. Questionnaire Survey Results**

## Appendix E

Table 12.15 - Statistical Summary of Quantitative Questions

<i>Questions</i>	<i>Count</i>	<i>Mean</i>	<i>Std</i>	<i>Min.</i>	<i>25%</i>	<i>50%</i>	<i>75%</i>	<i>Max</i>
<i>The GenAI4RSL was easy to use.</i>	13	4,46	0,66	3	4	5	5	5
<i>The instructions provided by GenAI4RSL were clear and understandable.</i>	13	3,92	0,64	3	4	4	4	5
<i>The interface of GenAI4RSL was intuitive.</i>	13	4,31	0,63	3	4	4	5	5
<i>How would you rate the ability of GenAI4RSL to understand and process requirements specified in natural language?</i>	13	3,69	0,75	3	3	4	4	5
<i>I found the suggestions provided by GenAI4RSL useful for improving the specification of requirements.</i>	13	3,62	0,87	2	3	4	4	5
<i>I was able to follow all the instructions in the guide without difficulties.</i>	13	4,69	0,48	4	4	5	5	5
<i>The conversion tasks were clearly defined.</i>	13	4,77	0,44	4	5	5	5	5
<i>GenAI4RSL was accurate in converting NL to RSL.</i>	13	3,54	0,78	3	3	3	4	5
<i>GenAI4RSL was accurate in converting RSL to NL.</i>	13	3,77	0,73	2	4	4	4	5
<i>The conversion from NL to RSL by GenAI4RSL maintained the logic and format as specified in the document "RSL2_GenAI4RSL_CG_RefinedSpecification_Revenue Items".</i>	13	3,69	1,11	2	3	4	5	5
<i>The result of the conversion from RSL to NL by GenAI4RSL includes all necessary functional and non-functional requirements.</i>	13	3,62	1,12	1	3	4	4	5

*The result of the conversion from RSL to NL by GenAI4RSL provides sufficient details for me to understand the objective or perform my tasks without needing additional clarifications.*

*The requirements resulting from the conversion from RSL to NL by GenAI4RSL are clearly and unequivocally stated.*

*The language used in the result of the conversion from RSL to NL by GenAI4RSL is consistent. (The requirements do not contradict each other and use uniform terminology throughout the document.)*

*The result of the conversion from RSL to NL by GenAI4RSL is organized and formatted in a way that is understandable for the user or other stakeholders.*

*The result of the conversion from RSL to NL by GenAI4RSL presents greater clarity and precision of specification compared to the original document "NLI\_CG\_Original Specification\_Requirements\_Revenue\_Items".*

*Based on this experience, I believe that the use of Artificial Intelligence tools could have a positive impact on the requirements specification process.*

13	3,15	1,07	2	2	3	4	5
13	3,85	0,9	2	4	4	4	5
13	4,23	0,6	3	4	4	5	5
13	4,23	0,73	3	4	4	5	5
13	3,08	1,38	1	2	3	4	5
13	3,92	0,64	3	4	4	4	5