Towards an Integrative Approach Between the SofIA Methodology and ChatGPT for the Extraction of Requirements from User Interviews

P. Peña-Fernández, I. Ruiz-Marchueta and J. A. García-García[a] and M. J. Escalona Cuaresma[b]
University of Seville, Avda. Reina Mercedes s/n, Seville, Spain

Keywords: Software Requirements, ChatGPT Integration, SofIA Methodology, Semi-Automated Approach, Software Engineering.

Abstract: The elicitation and specification of software requirements are critical activities in software engineering, usually involving interviews between analysts and end users. These interactions are essential for understanding user needs but can lead to inconsistencies or incomplete information in the subsequent generation of use cases. This paper explores the integration of ChatGPT with the SofIA software methodology to address these challenges, leveraging natural language processing capabilities to enhance the transformation of interview transcripts into detailed use cases. The proposed approach combines the structured guidance of SofIA with ChatGPT's ability to process and generate coherent textual outputs, facilitating the automated identification, categorisation, and refinement of requirements. A proof of concept in a real-world software development scenario was conducted to evaluate this integration, focusing on metrics such as accuracy, completeness, and time efficiency. This work contributes to the advancement of requirements engineering by introducing a semi-automated, user-centred approach that bridges the gap between human interviews and formal documentation. Future research directions include scaling the approach to more complex domains and refining its adaptability to diverse project requirements.

1 INTRODUCTION

Requirements engineering is the field of engineering that focuses on defining the real-world goals, functions, and constraints of systems (Laplante and Kassab, 2022). It also deals with how these elements relate to detailed system specifications and how they evolve over time and across related system families. This process is the pillar that ensures that a final product meets the objectives for which it was designed. Requirements engineering is concerned not only with gathering needs, but also with analising them, prioritising them, and transforming them into clear specifications that serve as the basis for design and implementation.

Talking to customers and end-users is an essential practice during requirements capture, as they have direct insight into the problems the system needs to solve or the opportunities it needs to exploit. This dialogue helps developers understand user expectations and the constraints of the environment in which the system will operate. This exchange of information not only enriches the understanding of the project, but also encourages collaboration and a sense of shared commitment, which can be crucial to the success of development (Pohl, 2016).

However, understanding the needs of customers and end-users is not without its challenges. One of the main challenges lies in communication. Customers often express their requirements in a vague or imprecise manner, using ambiguous terms that can be interpreted in different ways. This is compounded by the possibility that end-users may not have a clear technical vision, making it difficult to translate their needs into concrete specifications (Nuseibeh and Easterbrook, 2000). Additionally, stakeholders may have conflicting goals or interests, making it hard to prioritize requirements effectively. Another major challenge is resistance to change; some people may be hesitant to accept new technological solutions for fear of disrupting their usual way of working. Finally, requirements may evolve as the project progresses, requiring continuous and flexible management to avoid deviations that compromise the ultimate goal.

In this context, artificial intelligence (AI) and

[a] https://orcid.org/0000-0003-2680-1327
[b] https://orcid.org/0000-0002-6435-1497

Peña-Fernández, P., Ruiz-Marchueta, I., García-García, J. A. and Cuaresma, M. J. E.
Towards an Integrative Approach Between the SofIA Methodology and ChatGPT for the Extraction of Requirements from User Interviews.
DOI: 10.5220/0013707900003985

In *Proceedings of the 21st International Conference on Web Information Systems and Technologies (WEBIST 2025)*, pages 165-172
ISBN: 978-989-758-772-6; ISSN: 2184-3252

large-scale language models (LLMs), such as ChatGPT, are emerging as valuable tools in requirements engineering. These technologies have the potential to address some of the discipline's most complex challenges by facilitating both the capture and analysis of user needs. For example, a model such as ChatGPT can quickly process large volume of text, identifying patterns and recurring themes that reflect the most important needs. This is especially useful in contexts where human analysts face time or resource constraints. In addition, through natural language processing, these tools can identify inconsistencies, redundancies, or contradictions in requirements documents (Ferrari et al., 2016).

Although ChatGPT can enhance the way user requirements are captured in software projects, it is necessary to integrate this technology within the procedures of software engineering methodologies.

Due to this, the contribution of this paper is the expansion of the SofIA (Software Methodology for Industrial Applications) methodology (Escalona et al., 2023) to integrate ChatGPT API for the automated generation of use case diagrams. This integration allows analysts to focus fully on the client during interviews without needing to take extensive notes throughout the conversation, making the client feel more heard. In terms of requirements engineering, this approach improves quality and saves valuable time for the analyst.

Finally, this paper is structured as follows. After this introduction, Section 2 describes some related works. Section 3 presents the materials and and theoretical and technological foundations on which our proposal is based. Section 4 and Section 5 describe our results and an initial discussion of test cases, respectively. Finally, Section 6 states final conclusions and some future works.

2 RELATED WORKS

Recent advancements and updates in Large Language Models (LLMs) have been integrated into numerous projects within the context of software development, aiming to save time and improve quality. In particular, we have explored various proposals that assess the usefulness of LLMs in requirements engineering, especially for modelling and automating these processes.

On one hand, Ferrari *et al.* (Ferrari et al., 2024) conducted an experiment to evaluate the reliability of ChatGPT in generating UML sequence diagrams. They used 20 industrial requirements documents from real-world projects and crafted a prompt that included the request, the list of requirements, and the desired output format. The results produced by ChatGPT were then visualised using PlantText. The evaluation by experienced engineers showed promising outcomes in terms of comprehensibility, adherence to standards, and terminological consistency, although shortcomings were noted in terms of completeness and correctness. Nevertheless, it is believed that with additional contextual information and deeper domain knowledge, ChatGPT's model generation could significantly improve.

Similarly, Fill *et al.* (Fill et al., 2023) carried out various experiments using ChatGPT, based on the latest GPT-4 models, to investigate its application in generating and interpreting conceptual models. In their tests, they explored the creation of Entity-Relationship (ER) diagrams, Business Process diagrams, and UML class diagrams, designing specific prompts for each case. These prompts first provided a brief context and then defined the task. In some experiments, they used few-shot learning by providing examples to ChatGPT, while in others they employed a zero-shot approach, often defining the output structure in JSON format. The results demonstrated the potential of LLMs to assist in modelling tasks, as ChatGPT strictly adhered to the custom formats specified in the prompts.

On the other hand, Ben (Ben Chaaben, 2024) developed a tool based on LLMs that integrates with a modelling environment to provide continuous support throughout the modelling process, leveraging GPT-3. He used few-shot learning to enhance model comprehension and focused his study on class and activity diagrams. His tool offers three modes of assistance: an automatic mode that provides real-time suggestions, an on-demand mode where users can request specific recommendations, and a final review mode offering suggestions for possible model improvements. It was tested by 30 participants who expressed satisfaction with the experience. Test results indicate that the assistance significantly impacts modeling productivity, contribution, and creativity, demonstrating that these models can not only offer useful suggestions but also understand and construct models effectively. This opens the door to the development of even more advanced tools capable of autonomously generating models from natural language specifications.

In the same vein, Bajaj *et al.* (Bajaj et al., 2022) proposed MUCE (Multilingual Use Case model Extractor), a tool designed to extract use case models from functional requirements written in plain text in any language. In addition to automatically generating actors and use cases, the tool provides a validation mechanism. However, it has a significant limita-

tion: the analyst must have previously written the requirements document, as the text must be copied and pasted into the tool. Furthermore, MUCE does not generate use case or class diagrams, which reduces its usefulness in an automated workflow.

Another approach was presented by Herwanto (Herwanto, 2024), who explored the generation of Data Flow Diagrams (DFDs) using ChatGPT. The experiment was based on a structured prompt composed of four parts: task description, detailed instructions, input consisting of user stories, and an example using few-shot learning. The generated output included a CSV-formatted syntax, which was then imported into draw.io for visualization and editing of the diagram. Although the study confirms the potential of ChatGPT to assist in visual modeling within software engineering, the process still requires significant manual intervention. The development team must first define the user stories, and the tool is not integrated with any specific methodology, making the process more labor-intensive.

While the potential benefits of using ChatGPT in requirements engineering are clear, there is still significant room for exploration and improvement. Our goal is to leverage this technology to automate the requirements engineering process as much as possible, covering everything from stakeholder interviews to model generation. We believe that combining LLMs with the SofIA methodology and effective prompt engineering will allow us to develop a more precise and autonomous tool, reducing the engineers' workload and optimising the transition from requirements analysis to modeling.

3 MATERIALS AND METHODS

OpenAI provides accessible and well-documented APIs that allow developers to integrate large language models into custom applications, services, or systems. These APIs work similarly to ChatGPT (OpenAI, 2024b), where natural language prompts are sent to the model, which then generates context-based responses.

The API offers access to various models designed for different use cases. In this project, we will specifically work with the o4-mini model, the latest small model in OpenAI's "o" series. It is optimised for fast and effective reasoning, with exceptional performance in programming and visual tasks (OpenAI, 2024a). Despite being smaller than other models, it provides an excellent balance of cost, performance, and versatility.

To use the API effectively, it is essential to understand the concept of tokens, the smallest units of text the model processes. Tokens are used both in the input (prompt) and the output (response), meaning that cost and performance are directly tied to token usage. Optimizing prompts and managing token counts is crucial for efficiency and cost reduction.

Before integrating OpenAI's APIs into any software methodology, developers must obtain an API key. This key functions as a unique identifier that authenticates requests, controls access, and logs usage (Auger and Saroyan, 2024).

Access to the API is managed through a pay-as-you-go pricing model, where the total cost depends on the selected model and the number of tokens used for input and output. OpenAI regularly publishes its pricing on its official website, allowing developers to manage budgets, set usage limits, and plan resource consumption efficiently.

During the development of this project and the execution of the tests, approximately 100.000 tokens were used, resulting in a total cost of only $0.02 This highlights the remarkable cost-efficiency of OpenAI's o4-mini model. Despite the intensive use of the model to generate and process complex information, the economic impact was negligible. This demonstrates one of the key advantages of using optimised language models: the ability to integrate AI into software engineering processes without incurring high costs an especially valuable benefit for academic, research-based, or budget-constrained projects.

The SofIA methodology (Software Methodology for Industrial Applications) was developed by the ES3 research group (Engineering and Science for Software Systems) at the University of Seville. It is a model-based methodological framework supported by a CASE tool also named SofIA. This methodology originates from a previous proposal, NDT (Navigational Development Techniques), and its associated tool, NDT-Suite (García-García et al., 2014). The creation of SofIA stems from over two decades of experience applying NDT-Suite in industrial contexts.

The SofIA tool was developed by extending functionalities from platforms such as Enterprise Architect (Sparx Systems, 2022) and Draw.io, resulting in a powerful CASE tool for software application design and development. It provides high flexibility in the design starting points and offers automated support for bidirectional traceability, which is crucial for software lifecycle management. SofIA focuses particularly on the early stages of development, including requirements engineering, prototyping, use cases, and data structure modeling (Escalona et al., 2023).

SofIA is defined as an MDE (Model-Driven Engineering) methodology (Escalona et al., 2023), al-

lowing comprehensive coverage of the entire software lifecycle—from feasibility studies to maintenance. However, its main strength lies in the requirements phase.

In this work, SofIA will also serve as the base methodology, but with an innovative proposal: the integration of artificial intelligence functionality via the OpenAI API. The goal is to automate key requirements engineering processes, particularly the automatic generation of use cases based on content extracted from elicitation interviews between the requirements analyst and the client.

4 RESULTS

This section describes in detail our proposal to integrate ChatGPT API for the generation of use cases and class diagrams in SofIA. For this purpose, we have divided it into two subsections to clearly explain how these materials and methods, explained in the previous section, work. Section 4.1 describes the technological and functional architecture of our proposal and Section 4.2 describes the communication flow once our proposal begins to be used.

4.1 Technological and Functional Architecture of Our Proposal

The architecture of the proposed approach is divided into three main phases, as shown in Figure 1: (Phase 1) Requirement Gathering Phase, during which an interview is conducted between the client and the analyst to identify and collect the needs, expectations, and objectives of the desired system or product; (Phase 2) Requirement Modeling Phase, which involves generating, as a final product, model diagrams that clearly and accurately describe and represent the requirements gathered in the previous phase; and (Phase 3) Intelligent Data Process Phase, which aims to extract structured information about the requirements from the recorded interview, generating a file that will be used by SofIA to create the diagrams.

First, during Phase 1, the interview between the client and the analyst is (Figure 1.a) recorded in order to later obtain an MP3 audio file (Figure 1.b) that captures the entire conversation, including the client's expectations, as well as the recommendations and modifications discussed between both parties to ensure the greatest completeness of the requirements.

Phase 2, developed in the Enterprise Architect tool, is composed of four modules that operate sequentially. It begins with the Data Transformation Module, where the MP3 file is converted into transcribed text, generating a PDF file that contains the complete conversation. The PDF format is chosen to ensure proper readability by the ChatGPT API.

This PDF file serves as input for the API Cloud Module, corresponding to Phase 3, which is responsible for generating the model. This module is a cloud environment that hosts ChatGPT's artificial intelligence services, allowing access to its capabilities through the API we have integrated into the SofIA methodology. The PDF file is analyzed in the cloud by ChatGPT (Figure 1.c), and, following the prompt defined in the source code, a JSON file (Figure 1.d) is generated and returned to SofIA to continue the modeling process.

Back in phase 2, the Integration Module receives the JSON file generated by ChatGPT after analyzing the PDF. This JSON file consists of a numbered list of requirement-derived objects, organized according to the structure defined in the prompt, which enables SofIA to properly interpret them.

Next, the Transformation Module takes the JSON file as input to generate the model requested by the analyst (use cases diagrams) following the methodology defined by SofIA.

Finally, the Definition Module allows for the visualization of the generated modeling diagrams, facilitating their review and editing by the analyst.

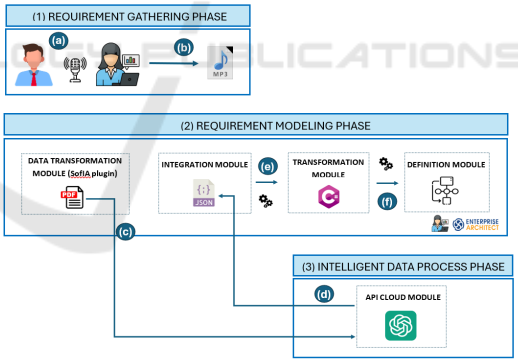

Figure 1: Technological and functional architecture.

4.2 Communication Flow

The communication flow of our project proposal will be reflected in a sequence diagram, as shown in Figure 2. As we have seen in the architecture defined in the section before, the proposal includes two actors who are responsible for initiating the interaction.

Firstly, client and analyst's first and only interaction is `<<start interview>>` which, as we already know, is recorded. The analyst's second interaction is `<<generate transcription>>`, which

involves converting the recorded conversation into a PDF file.This process is carried out in the "Data Transformation Module".

Before proceeding with the use of the generated PDF, it is important to mention that the analyst must configure the integration with the ChatGPT API. To do this, they will need to log in, and if successful, they will be able to use the API without any restrictions. Next, the PDF file is uploaded to the "Integration Module", which instantly interacts with the cloud module of the API using the `<<generate object List>>` feature. This interaction returns a JSON file containing a list of objects.

This JSON file can be reviewed and/or edited by the analyst if they consider it necessary. Once reviewed, `<<get model diagrams>>` is the next interaction performed by the analyst. The SofIA methodology generates the model diagrams in the "Transformation Module."

Finally, the process is completed when the "Definition Module" returns the obtained model diagrams to the analyst, enabling them to begin working.

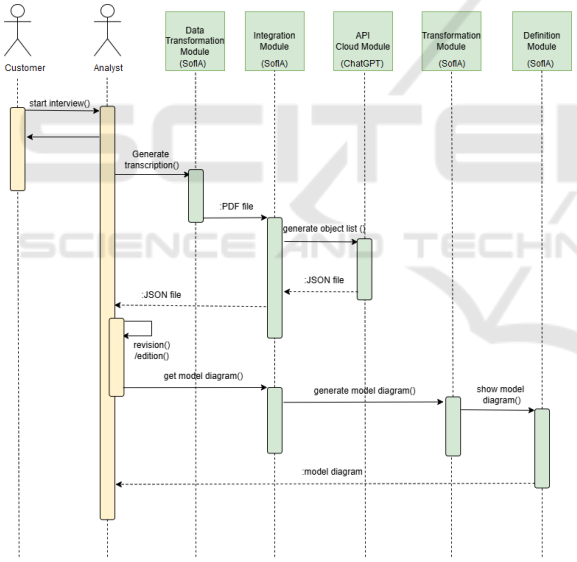

Figure 2: Communication Flow of our proposal.

5 TEST CASE AND DISCUSSION

During the development of our proposal, it has been necessary to perform a thorough prompt engineering task to maximise the accuracy of our results and their integration with SofIA. Prompt engineering is a key practice to maximise the potential of language models (e.g. ChatGPT) and its aim is to design and refine the instructions provided to the model, ensuring that they are clear, specific and target-oriented. This constant creation and improvement of prompts not only optimises the interaction with the AI, but also minimises ambiguous interpretations and unexpected results.

The effectiveness of prompt engineering lies in the ability to experiment with different formulations, adjusting the level of detail, the context and the keywords used. Moreover, this practice allows the strengths of the model, such as creative content generation or complex analysis, to be exploited while mitigating its limitations. Some authors have studied these prompt engineering features and their advantages and challenges. For example, Arora *et al.* performance (Arora et al., 2022) highlight in their study the significance of explicitly defining the desired output within prompts to ensure responses are generated in the intended format. In the absence of such specification, ChatGPT autonomously determines the response structure. Implementing this descriptive constraint within prompts enhances effectiveness and optimises. Li (Li, 2023) also asserts in his study that zero-shot prompts can surpass few-shot prompts, highlighting their high interpretability and the absence of a need for prior training. These prompts adhere to a structured pattern to precisely define the expected output by incorporating explicit constraints. The use of zero-shot prompts necessitates the identification of the most optimized optimised formulation to efficiently execute the intended task. The developed prompt has the next structure: (1) an introduction of the role it's going to work as, (2) the task objective to give coherence to the prompt, (3) the output format specification and (4) the restrictions and constraints to get a high quality output.

Below is the prompt that has been defined for use in this proposal.

Prompt

You are an expert engineer in requirements analysis, object-oriented software design, and UML modeling. Your goal is to build a complete, coherent, and well-structured use case diagram based on a project conversation.
Your response must contain only the list of use cases in the following format, with no additional text: `(name#description#list of associated actors)`
Use one line break for each use case. Do not include anything else in your response.
Analyze the following conversation about the creation of a web project — `fileText` — and extract the use cases present in the dialogue.

Prompt: First part of the prompt.

Prompt

Generate a structured list format, ensuring each use case is well-defined and reflects the requirements mentioned in the conversation. Each use case should be represented as a tuple with the following attributes:

- **Name:** A brief and descriptive title of the use case.
- **Description:** A clear summary of the functionality or purpose of the use case.
- **List of associated actors:** Users or systems participating in the use case, separated by commas.

The output format must be the following: `(name#description#list of associated actors)` Additional rules:

- Do not include quotation marks or extra characters in actor names.
- Separate each use case with a line break.
- Ensure the description is concise yet complete.
- Extract only relevant use cases, avoiding irrelevant or redundant information.

Expected output example:

```
(User registration#Allows a new user to create an account in the system#User)
(Login#The user enters credentials to access the platform#User)
(Project management#Users can create, modify and delete projects#Administrator)
(Configuration update#Modifies system parameters based on permissions#System)
(Send notification#Sends a notification to the appropriate user#System)
```

Now, process the conversation and generate the list of use cases following these instructions.

Prompt: Second part of the prompt.

In order to validate the conceptual utility of the tool within a real-world context, a test was conducted involving the generation of use cases from the transcript of an actual client interview. To this end, a business professional—the manager of a massage therapy establishment—was contacted. This individual expressed the need for a tool to assist in automating certain aspects of their workflow.

The interview was conducted via Microsoft Teams, lasting approximately 15 minutes, with both analysts from this project participating. The conversation was automatically transcribed using Teams' built-in transcription functionality, resulting in a PDF file. This file was subsequently used as input for the tool, enabling ChatGPT to autonomously generate both the use case diagrams and the corresponding class diagrams.

The generation of the diagram required approximately 30 seconds and did not involve any manual intervention. It is important to emphasize that no modifications or adjustments were made to the output produced by ChatGPT, in order to objectively assess the system's capacity to extract meaningful and conclusive information from the content of the conversation.

In Figure 3 it is displayed the resulting use case diagram. Although no fields were modified in this instance, the system allows manual editing of use case names in situations where ambiguity may arise, enabling the user to replace them with more suitable alternatives. Similarly, additional details or extended descriptions may be manually incorporated if a more comprehensive explanation of a given use case is desired.

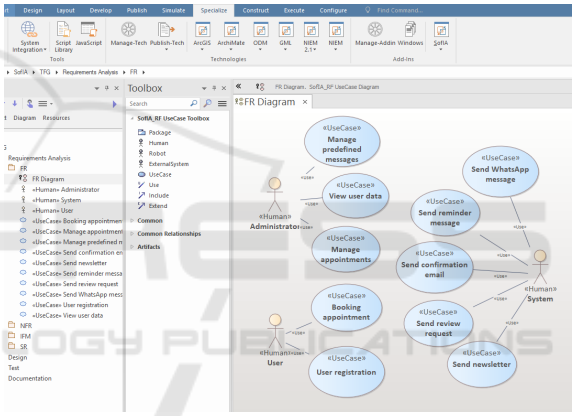

Figure 3: Use case diagram.

Before reviewing the results generated by the tool, a junior analyst conducted their own analysis and review of the interview to manually extract the use case diagram based on the client's system requirements. The result of the analyst's work can be seen in Figure 4.

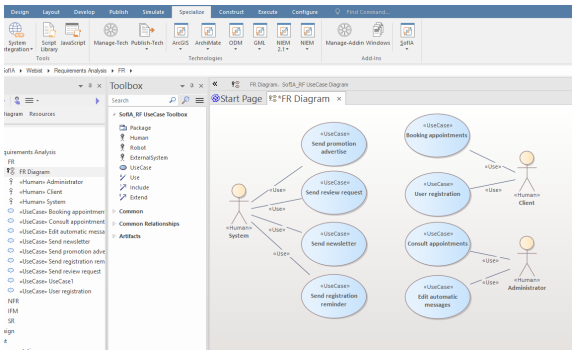

Figure 4: Analyst´s use case diagram.

Upon initial inspection, it can be seen that the tool identified the same actors as the analyst; however, the tool was able to extract more use cases. Upon carefully reviewing the interview (located at the end of the paper, in a hyperlink within the appendix) and analyzing the result, it can be concluded that the two additional use cases extracted by SofIA+ChatGPT are indeed necessary.

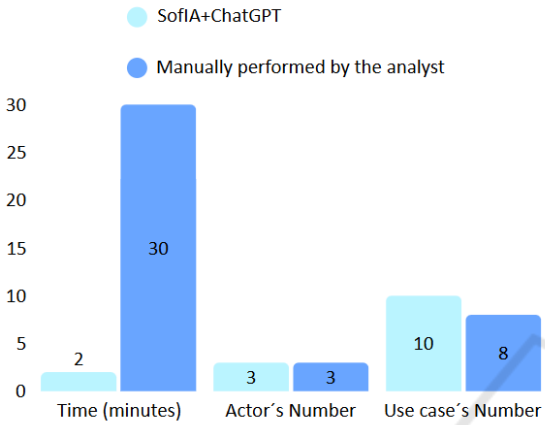

Figure 5: Comparative chart.

As a conclusion of this validation test, it is demonstrated that the integration of ChatGPT into the SofIA methodology has been capable of efficiently generating the use cases for a system requested by a real client, with a high degree of similarity to the work produced by a human analyst. The most noteworthy aspects are the speed of the process and its usefulness as a support tool in requirements engineering. However, the success of the system in more complex scenarios or in situations where requirements are incomplete or contradictory still needs to be studied.

6 CONCLUSIONS

This paper explores the integration of the ChatGPT with the SofIA methodology to enhance the elicitation and specification of requirements in software engineering. In this context, interviews with clients and end-users are essential for understanding their needs, although challenges such as ambiguity, inconsistencies, and evolving requirements often arise. To address these limitations, the research proposes a semi-automated solution that combines ChatGPT's natural language processing capabilities with SofIA's flexible structure.

The main contribution lies in the automation of processes, such as the generation of use cases, derived from interview transcripts. This approach allows analysts to focus on clients during conversations, avoiding distractions related to extensive note-taking. Additionally, a workflow was implemented to convert recorded interviews into high-quality UML models, improving efficiency and precision in requirements specification. This was achieved through an iterative design of optimized prompts, ensuring outputs aligned with SofIA's standards.

The study's findings were validated through test cases, demonstrating the approach's effectiveness in generating models and filtering irrelevant information. Moreover, benefits such as reduced analysis time and the extraction of high-quality documented requirements.

It should be noted that the results obtained through this tool must be validated by an analyst, as ChatGPT may not generate the use cases with complete accuracy or in the most appropriate way for the system's context. It is important to emphasize the rapid growth and evolution of large language models (LLMs) in recent years, and how their ability to understand natural language will continue to improve. This will allow them to define requirements in a way that more closely resembles human reasoning, reducing the need for post-editing even more so than with our current tool.

Despite the positive results obtained, certain limitations in the validation process must be acknowledged. For instance, it would be necessary to compare the tool's performance with a more experienced analyst in order to clearly identify its strengths and areas for improvement. Furthermore, it is important to note that the tool relies on a carefully crafted prompt; any significant modifications or reductions to this prompt could lead to less satisfactory results than those achieved in this study.

Finally, as future works, it is proposed to expand the system's functionalities, such as the ability to generate class diagram and user stories from interview transcripts. In addition, it is planned to conduct validation tests in more domains to evaluate the system's capability to understand complex fields, such as the healthcare, financial, or legal sectors. The proposal also aims to scale to projects with more diverse requirements and more complex structures, adapting the system to a broader variety of contexts. Another idea being considered is the integration of an active listening API, which would allow real-time capture and transcription of conversations between the client and analyst, eliminating the need to use external tools or manually upload files from the file browser. This would enable greater automation and fluidity in the requirements capture and analysis process.

REFERENCES

Arora, S., Narayan, A., Chen, M. F., Orr, L., Guha, N., Bhatia, K., Chami, I., Sala, F., and Ré, C. (2022). Ask me anything: A simple strategy for prompting language models. *arXiv preprint arXiv:2210.02441*.

Auger, T. and Saroyan, E. (2024). Overview of the openai apis. In *Generative AI for Web Development: Building Web Applications Powered by OpenAI APIs and Next. js*, pages 87–116. Springer.

Bajaj, D., Goel, A., Gupta, S., and Batra, H. (2022). Muce: a multilingual use case model extractor using gpt-3. *International Journal of Information Technology*, 14(3):1543–1554.

Ben Chaaben, M. (2024). Software modeling assistance with large language models. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, pages 188–191.

Escalona, M.-J., García-Borgoñon, L., García-García, J., López-Nicolás, G., and de Koch, N. P. (2023). Choose your preferred life cycle and sofia will do the rest. In *International Conference on Web Engineering*, pages 359–362. Springer.

Ferrari, A., Abualhaijal, S., and Arora, C. (2024). Model generation with llms: From requirements to uml sequence diagrams. In *2024 IEEE 32nd International Requirements Engineering Conference Workshops (REW)*, pages 291–300. IEEE.

Ferrari, A., Spoletini, P., and Gnesi, S. (2016). Ambiguity and tacit knowledge in requirements elicitation interviews. *Requirements Engineering*, 21(3):333–355.

Fill, H.-G., Fettke, P., and Köpke, J. (2023). Conceptual modeling and large language models: impressions from first experiments with chatgpt. *Enterprise Modelling and Information Systems Architectures (EMISAJ)*, 18:1–15.

García-García, J. A., Escalona, M. J., Domínguez-Mayo, F. J., Salido, A., et al. (2014). Ndt-suite: a methodological tool solution in the model-driven engineering paradigm. *Journal of Software Engineering and Applications*, 7(04):206.

Herwanto, G. B. (2024). Automating data flow diagram generation from user stories using large language models. In *7th Workshop on Natural Language Processing for Requirements Engineering*.

Laplante, P. A. and Kassab, M. (2022). *Requirements engineering for software and systems*. Auerbach Publications.

Li, Y. (2023). A practical survey on zero-shot prompt design for in-context learning. *arXiv preprint arXiv:2309.13205*.

Nuseibeh, B. and Easterbrook, S. (2000). Requirements engineering: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 35–46.

OpenAI (2024a). OpenAI. Último acceso: 7 de abril de 2025.

OpenAI (2024b). Text generation with the openai api. Último acceso: 7 de abril de 2025.

Pohl, K. (2016). *Requirements engineering fundamentals: a study guide for the certified professional for requirements engineering exam-foundation level-IREB compliant*. Rocky Nook, Inc.

APPENDIX

In Section 5, reference is made to the interview conducted between the client and the junior analyst, the content of which was automatically transcribed and used for the generation of the corresponding use cases and diagrams. The full transcript of this interview, carried out via Microsoft Teams, is available as an external complementary document for consultation. It can be accessed at the following link:Interview