

A Conceptual Framework for the Generation of High-Quality Software Requirements

Mauro José Pacchiotti¹, Luciana Ballejos¹ y Mariel Ale¹

¹ Centro de I+D de Ingeniería en Sistemas de Información

UTN-FRSF

{mpacchiotti, lballejos, male}@frsf.utm.edu.ar

Abstract. Requirements expressed in natural language are an essential artifact in the software development process, as all stakeholders can understand them. However, their inherent ambiguity remains a persistent challenge. To address this issue, organizations such as the Institute of Electrical and Electronics Engineers (IEEE) and the International Council on Systems Engineering (INCOSE) publish guidelines with rules that support the task of writing clearer requirements. Additionally, agile methodologies propose patterns and structures for formulating stakeholder needs in natural language, aiming to reduce ambiguity. Nevertheless, differences in stakeholders' understanding of the requirements and how to express them correctly make the specification task even more difficult. Recently, large language models (LLMs) have emerged to enhance natural language processing tasks. These are deep learning architectures that emulate attention mechanisms similar to those used by humans. This work aims to assess the potential of LLMs in this domain. The goal is to use these models to improve the quality of software requirements, assisting analysts in the tasks of analysis and specification. The proposed framework, its architecture, key components, and their interactions are described. A conceptual test is also conducted to evaluate the framework's usefulness. Finally, the paper discusses the framework's potential and limitations, as well as future directions for its validation and refinement.

Keywords: Requirements Engineering, Requirements Quality, Large Language Models.

Un marco conceptual para la generación de requerimientos de software de calidad

Mauro José Pacchiotti¹, Luciana Ballejos¹ y Mariel Ale¹

¹ Centro de I+D de Ingeniería en Sistemas de Información

UTN-FRSF

{mpacchiotti, lballejos, male}@frsf.utn.edu.ar

Resumen. Los requerimientos expresados en lenguaje natural son un artefacto indispensable en el proceso de desarrollo de software, ya que pueden ser comprendidos por todas las partes interesadas. Sin embargo, su ambigüedad representa un desafío persistente. Para abordar esta problemática, organizaciones como el Institute of Electrical and Electronics Engineers (IEEE) y el International Council on Systems Engineering (INCOSE) publican guías para escribir requerimientos con reglas que aportan a la tarea. Por otro lado, las metodologías ágiles proponen patrones y estructuras para redactar en lenguaje natural las necesidades de las partes interesadas, tratando de evitar la ambigüedad. Sin embargo, la diferencia de conocimiento entre estas partes respecto a los requerimientos y su correcta formulación dificulta la tarea de especificación. Recientemente, los grandes modelos lingüísticos (LLM, por sus siglas en inglés) han emergido para mejorar las tareas de procesamiento de lenguaje natural, estos son arquitecturas basadas en modelos profundos que emulan mecanismos de atención similares a los humanos. Este trabajo tiene como objetivo evaluar el potencial de los LLM en este dominio. El propósito es utilizar estos modelos para mejorar la calidad de los requerimientos de software, ayudando a los analistas en las tareas de análisis y especificación de requerimientos. Se detallan el marco propuesto, la arquitectura, los componentes clave y las interacciones entre ellos. Además, se desarrolla una prueba conceptual de la propuesta para evaluar su utilidad. Por último, se discuten el potencial y las limitaciones del marco de trabajo, así como las direcciones futuras para continuar su validación y refinamiento.

Palabras clave: Ingeniería de Requerimientos, Calidad de Requerimientos, Grandes Modelos Lingüísticos.

1 Introducción

El Proceso de Ingeniería de Requerimientos (IR) es esencial en el desarrollo de software. El objetivo de este proceso es garantizar que se incorporan todas las necesidades, limitaciones y exigencias de calidad del cliente, y que el software a producir las satisface. Los requerimientos generados en esta fase proceden de varios tipos de fuentes: distintos documentos del dominio, stakeholders y sistemas heredados, si exis-

ten (Wiegers et al., 2013). Dos de estas fuentes, los documentos y los stakeholders, tienen como característica principal el uso del lenguaje natural. Por un lado, los documentos están escritos en lenguaje natural y por otro, el lenguaje natural facilita la comunicación y validación de los requerimientos con todas las partes interesadas.

El lenguaje natural tiene características no deseadas, tales como la ambigüedad, que dificultan la redacción del documento resultante del proceso de IR, llamado Especificación de Requerimientos de Software (ERS). Esta especificación contiene todas las necesidades, restricciones y demandas de calidad obtenidas de las partes interesadas y extraídas de los documentos y sistemas heredados. Un uso incorrecto del lenguaje natural en el proceso y en la creación de este documento, puede dar lugar a malentendidos u omisiones que dificulten el desarrollo del software en etapas posteriores.

INCOSE publica y mantiene una guía para escribir requerimientos (GtWR Homepage, 2025) que recomienda un conjunto de características y reglas para ayudar a los analistas en el proceso de especificación de requerimientos. En esta guía, se enumera un conjunto de características no deseadas, y la ambigüedad es la más común de ellas. La ambigüedad existe con frecuencia en las expresiones del lenguaje natural en varios sentidos, y es necesario eliminarla de las expresiones de requerimientos, porque éstas necesitan tener un significado único.

En los últimos tiempos, con el crecimiento de la inteligencia artificial, se han propuesto nuevos modelos profundos que han demostrado un gran rendimiento en el procesamiento del lenguaje natural. Estos modelos, llamados LLM, se basan en un modelo base propuesto en el trabajo *Attention is all that you need* (Vaswani et al., 2017), llamado Transformer. A partir del Transformer han surgido varios modelos con variantes de su arquitectura, algunos de los cuales han demostrado mejoras significativas en tareas de lenguaje natural, tales como la generación de resúmenes (Liu et al., 2024), traducciones (Volk et al., 2024) y respuestas a preguntas (Robinson et al., 2022), entre otras.

Este artículo pretende incorporar el uso de LLMs para ayudar al analista en tareas de especificación de requerimientos, basándose en las grandes cualidades de estos modelos para mejorar el procesamiento de lenguaje natural. Se propone un marco de trabajo para interactuar con los modelos en varias tareas dentro de un proceso que pretende mejorar la calidad de los requerimientos de software especificados.

El resto de este trabajo se organiza como sigue: la Sección 2 presenta los fundamentos, la Sección 3 describe el marco propuesto, la Sección 4 desarrolla una prueba conceptual de la propuesta y evalúa los resultados, y la Sección 5 discute los resultados, expone las conclusiones y propone los trabajos futuros.

2 Marco Teórico y Trabajos Relacionados

En tiempos recientes, con la llegada de los LLMs, distintos estudios han investigado y propuesto el uso de estos modelos profundos como herramienta en el proceso de IR. Entre ellos, Porter et al. (2025) revisan la evolución de las herramientas automáticas para evaluar la calidad. Destacan el potencial de la inteligencia artificial, especialmente los transformers, para realizar análisis más contextuales y precisos, y presentan la herramienta en desarrollo ARQM que integra estas tecnologías para mejorar la calidad de los requerimientos según el estándar IEEE 29148. También Reinpold et al.

(2024) investigan el uso de LLMs para verificar si las especificaciones técnicas cumplen con los requerimientos. Realizando pruebas con un conjunto de datos acotado y modelos avanzados como GPT-4o y Claude 3.5 Sonnet, obtuvieron puntuaciones F1 de entre el 79 % y el 94 % en la identificación de requerimientos no cumplidos. En otra línea Norheim et al. (2024) realizan una revisión y analizan críticamente el éxito y las dificultades de aplicar LLMs en distintas tareas del proceso de IR, resaltando y concluyendo la falta de una forma de evaluar el aporte real de estas herramientas a las tareas del proceso.

En particular, este artículo presenta un marco de trabajo, basado en el uso de LLMs, para mejorar la calidad de las expresiones de requerimientos de software.

2.1 Proceso de IR

La IR es una rama de la Ingeniería del Software (IS) que se ocupa de la gestión de los servicios y restricciones de los sistemas de software (Nuseibeh et al., 2000). El proceso de IR puede dividirse en cuatro actividades principales: estudio de factibilidad, obtención y análisis, especificación y validación de requerimientos, según Sommerville (Sommerville, 2016). Estas actividades no se ordenan secuencialmente, sino que pueden realizarse de forma iterativa a lo largo del proceso y a medida que cambian los requerimientos o se descubren nuevos. El marco propuesto en este trabajo pretende asistir a los analistas principalmente en las actividades de obtención, análisis y especificación de requerimientos.

2.2 Grandes Modelos Lingüísticos

A partir del trabajo *Attention is all that you need* (Vaswani et al., 2017), se propuso el Transformer, un modelo de aprendizaje profundo con una arquitectura dividida en dos estructuras, el codificador y el decodificador, basado en el mecanismo de atención multicabeza. Mientras que los modelos recurrentes procesan el texto secuencialmente, el Transformer utiliza un mecanismo de atención que le permite considerar todas las palabras de una frase simultáneamente. Este mecanismo permite a los modelos lingüísticos destacar las partes más relevantes de un texto de entrada para comprender mejor su significado y contexto. Así, el modelo puede captar las relaciones de largo alcance entre las palabras, lo que se traduce en un mejor rendimiento en diversas tareas de lenguaje natural.

Desde el Transformer, se siguieron desarrollando modelos basados en su estructura, tales como BERT (Bidirectional Encoder Representations from Transformers) (Devlin et al., 2018), GPT (Generative Pretraining Transformer) (Brown et al., 2020) y T5 (Text-to-Text Transfer Transformer) (Raffel et al., 2019), que se entrenaron con conjuntos de datos aún mayores, compuestos por texto y código. Estos modelos destacan por su capacidad para generar texto de alta calidad y realizar diversas tareas de procesamiento de lenguaje natural (PLN). Posteriormente, el uso de LLM se extiende aún más con la aparición de ChatGPT (ChatGPT Homepage, 2025) en 2022, una aplicación de la empresa OpenAI que permite utilizar IA generativa a través de una interfaz de chat basada en el modelo GPT para generar respuestas.

2.3 Prompt Engineering

La interacción con LLM generativos propone un nuevo paradigma en el que se puede interactuar con un modelo con expresiones en lenguaje natural. En estos modelos de lenguaje, la entrada del usuario se denomina *prompt*, que es una instrucción o conjunto de palabras diseñadas para iniciar y guiar la respuesta del modelo. Es importante la forma en que se genera esta expresión, ya que el modelo generará la salida a partir de la misma. Las buenas prácticas recomiendan elementos que un prompt puede contener en busca de lograr la respuesta esperada (Tabla 1).

Tabla 1. Elementos recomendados en un prompt. (Prompt Engineering Guide, 2024).

Elemento	Descripción
Instrucción	Tarea específica que se desea
Contexto	Información adicional que puede orientar al modelo y completar la respuesta.
Entrada	La entrada sobre la que se desea la acción.
Salida	Formato que se desea para la respuesta del modelo.

Los cuatro elementos en un prompt no son obligatorios. Incluirlos o no, depende de la necesidad expresada. Además, en la entrada que se le proporciona al modelo se pueden incluir ejemplos de la tarea que esperamos del mismo. Según la cantidad de ejemplos, se distinguen tres enfoques: zero-shot (sin ejemplos), one-shot (un solo ejemplo) y few-shots (varios ejemplos), siendo este último el más potente por su capacidad de generalización.

2.4 Generación Aumentada por Recuperación (RAG)

La arquitectura RAG (Figura 1) permite superar las limitaciones de conocimiento estático de los LLM, mejorando sus respuestas mediante la recuperación de información actualizada y contextual desde fuentes externas durante la generación.



Fig. 1. Proceso de RAG.

El proceso de RAG se inicia con el envío de un prompt por parte del usuario. Este es transformado en una representación vectorial y procesado por el recuperador, que consulta una base de datos externa, también vectorizada, con el fin de identificar los

fragmentos de información más relevantes. Esta base de datos almacena representaciones numéricas de textos que capturan su significado semántico, lo que permite recuperar información en función del contenido y no solo de coincidencias literales. Los textos recuperados se combinan posteriormente con el prompt original. Esta entrada enriquecida es procesada por un LLM que genera una respuesta contextualizada en función del contenido recuperado (Klesel y Wittmann, 2025).

3 Marco de trabajo propuesto

Esta propuesta se basa en interacciones entre requerimientos, fuentes de información y modelos con el objetivo de mejorar la calidad de los requerimientos. A continuación, se describe la arquitectura (Figura 2) y los distintos componentes que pueden implementarse dependiendo del dominio, alcance y tipo de problema a resolver.

3.1 Arquitectura

La arquitectura propuesta da soporte al proceso que tiene como objetivo mejorar la calidad de los requerimientos. Este proceso comienza con un primer modelo (Modelo evaluador) que evalúa la expresión de requerimiento. Esta evaluación puede implementarse con distintos tipos de LLM o ensambles, dependiendo de qué tan amplias se pretendan la evaluación y la salida. Puede utilizarse desde un simple codificador como parte de un clasificador binario, hasta un modelo generativo que devuelva una salida detallada describiendo la calidad del requerimiento. Es importante definir y documentar los criterios y modelo o modelos que se utilizarán en esta primera etapa, a fin de sustentar la decisión posterior de mejorar o no la expresión.

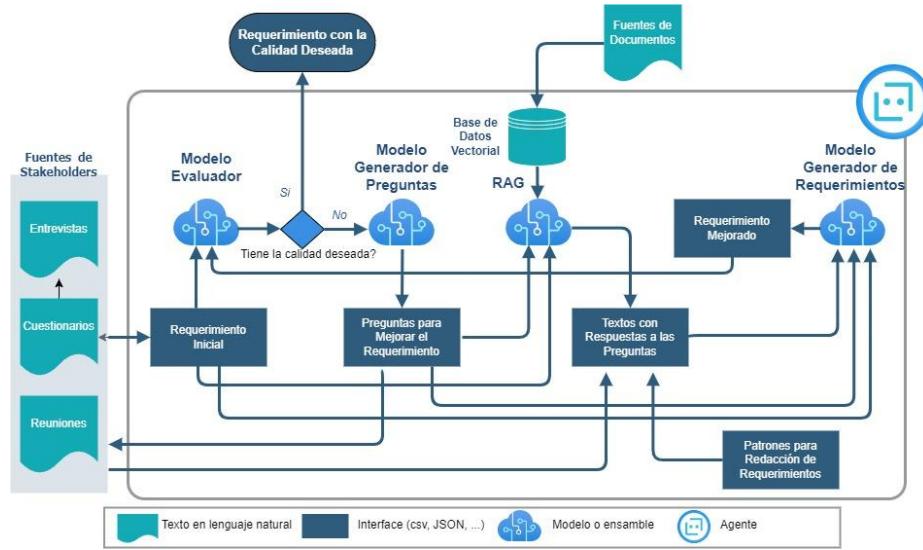


Fig. 2. Arquitectura de la propuesta.

El segundo modelo se ejecuta si, a partir de la salida del primero, se concluye que la calidad de la expresión del requerimiento no es suficiente. Para este caso, el segundo modelo (Modelo Generador de Preguntas) analiza la expresión del requerimiento y la

salida obtenida en la primera tarea, con el fin de obtener preguntas que mejoren su calidad.

Las preguntas generadas en la segunda tarea tienen como objetivo obtener respuestas que con su contenido aporten a mejorar la expresión del requerimiento, ya sea completando información o despejando dudas que disminuyan la ambigüedad de la expresión. Dependiendo del dominio, estas preguntas pueden utilizarse con dos tipos de fuentes. Por un lado, puede consultarse directamente a stakeholders y, por otro, pueden utilizarse con el tercer modelo propuesto y una fuente documental, si se dispone de ésta.

El tercer modelo es (RAG) toma como entradas un conjunto de preguntas sobre una expresión de requerimiento y busca en una base de datos vectorizada, con información de contexto, las posibles respuestas a estas preguntas. La utilización de este modelo depende de la existencia de documentación.

Un cuarto modelo (Modelo Generador de Requerimientos) recibe como entradas: el requerimiento original, un conjunto de preguntas y respuestas, fragmentos de contexto y patrones para redacción de requerimientos -si se dispone de éstos- para generar un nuevo requerimiento mejorado.

Finalmente, esta nueva expresión generada vuelve a ser evaluada por el primer modelo, a fin de verificar que cumple con la calidad esperada. De no ser así puede comenzarse una nueva iteración del proceso descrito.

3.2 Componentes

El funcionamiento del marco propuesto depende de tres componentes fundamentales: los modelos, las interfaces y, opcionalmente, un agente orquestador. Dado que se trata de un marco adaptable a diferentes contextos de aplicación, los modelos pueden ser de distintos tipos y estar implementados en diversas formas. Estos incluyen modelos generativos, clasificadores o ensambles orquestados, entre otros. Para automatizar las interacciones entre ellos, se requieren interfaces accesibles, que pueden variar desde simples archivos de texto hasta bases de datos documentales. Finalmente, si se busca una ejecución automática, un agente debe encargarse de interpretar interfaces, tomar decisiones y coordinar la ejecución. Aunque su uso no es obligatorio, su diseño resulta clave en implementaciones totalmente automatizadas.

3.3 Ejemplos de Implementación

A continuación, se muestran dos posibles ejemplos de implementación del marco con distintas adaptaciones de la arquitectura propuesta.

Ejemplo A (Figura 3). En esta implementación se utilizan tres modelos generativos. (a) es modelo preentrenado, ajustado, que evalúa la expresión, (b) es un modelo preentrenado, ajustado, que genera preguntas para mejorar la expresión y (c) es un modelo preentrenado y utilizado con una técnica de few-shot prompting que genera los requerimientos de acuerdo con una plantilla. En esta implementación no se utiliza un agente y es el usuario el que cumple este rol, primero decide si el requerimiento tiene la calidad deseada de acuerdo con la salida del modelo (a), y luego necesita recolectar de los stakeholders las respuestas a las preguntas generadas por el modelo (b), para finalmente darles el formato adecuado para que sean utilizadas en la última etapa.

Ejemplo B (Figura 4). Esta propuesta de implementación utiliza primero un clasificador (a) que evalúa desde distintas perspectivas la expresión para evaluar si ésta tiene la calidad deseada. Si no es así, un modelo generativo preentrenado y ajustado (b) genera las preguntas para mejorar la expresión, luego una arquitectura RAG provee las respuestas a las preguntas utilizando documentos de estándares y transcripciones de entrevistas y reuniones. Finalmente, un modelo generativo preentrenado utilizado con una técnica de few-shot prompting genera los requerimientos de acuerdo con una plantilla. En este caso, el proceso es totalmente automático por lo que se necesita un agente que orqueste el funcionamiento de la implementación.

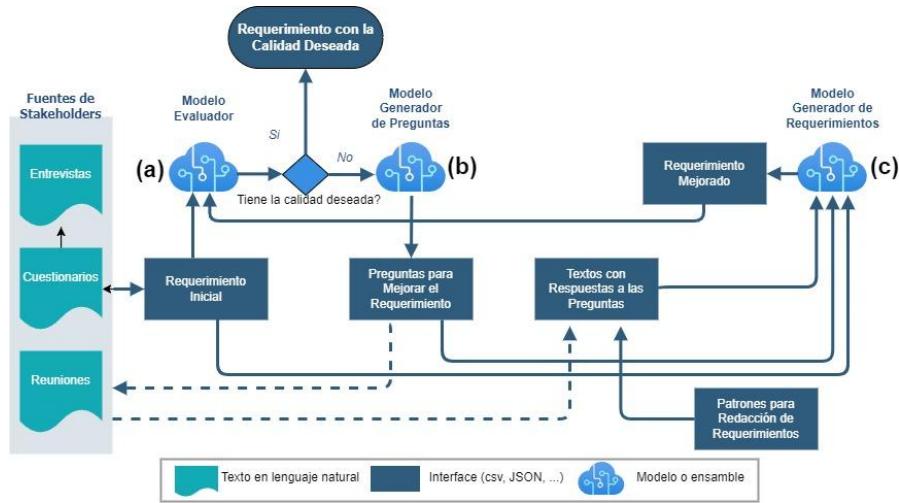


Fig. 3. Implementación del ejemplo A.

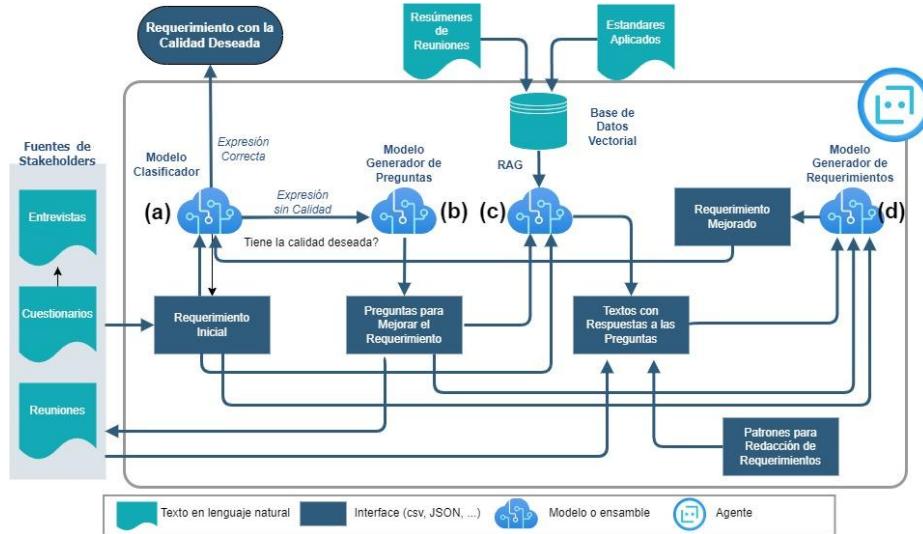


Fig. 4. Implementación del ejemplo B.

4 Prueba conceptual

4.1 Modelo utilizado

Es conocida la utilización de distintos asistentes, tales como Gemini (Gemini Homepage, 2025), Copilot (Copilot Homepage, 2025) o ChatGPT (ChatGPT Homepage, 2025), basados en LLM para dar soporte en infinidad de tareas, y no escapan a esta tendencia las actividades desarrolladas en el proceso de IR. Es por este motivo que las pruebas iniciales de esta propuesta se desarrollan utilizando un asistente bien conocido, ChatGPT versión 4o mini, aplicando técnicas de Prompt Engineering. Cabe aclarar que los resultados de las pruebas no son reproducibles con exactitud, debido a la naturaleza no determinística de las salidas que ofrecen estos modelos, aunque se repitieron algunas de las mismas obteniéndose resultados muy similares. Por último, el idioma elegido para el desarrollo de las pruebas es el inglés.

4.2 Metodología

Se utilizan tres requerimientos (Tabla 2) que fueron escritos para estas pruebas con el objetivo de representar características funcionales y no funcionales de forma comprensible. Primero, se consulta a ChatGPT con una consigna genérica para que sugiera mejoras a cada requerimiento por separado. Luego, se aplica una versión simplificada del marco propuesto con el mismo fin. Finalmente, se comparan los resultados para realizar una primera evaluación de la propuesta y planificar pruebas a mayor escala.

Tabla 2. Expresiones de Requerimientos utilizadas en las pruebas iniciales.

#	Expresión
R1	The system must allow the inventory manager to generate a list of missing products.
R2	The system will have a user-friendly interface and support all common browsers.
R3	A customer can cancel an order if he has not yet received it.

4.3 Evaluación genérica de las expresiones

Las tres expresiones de requerimientos fueron utilizadas para realizar una consulta genérica a un modelo que tiene como objetivo mejorar la calidad del requerimiento. Además, el prompt utilizado pretende obtener en la respuesta del modelo recomendaciones para mejorar la expresión y una nueva expresión mejorada.

Para la redacción de la consulta se siguieron las técnicas recomendadas de prompt engineering en busca de una mejor respuesta. Como se puede observar en la Figura 5, inicialmente la expresión llama a la acción de mejorar el requerimiento de software y referencia la expresión a ser examinada.

User Prompt

Indicate how the following software requirement can be improved.

Software Requirement: "The system must allow the inventory manager to generate a list of missing products."

Model
The given software requirement can be improved ...
Issues in the Original Requirement:
1. Ambiguity in "missing products" – Does this mean out-of-stock items...
Improved Requirement:
<i>"The system must enable the inventory manager to generate a report listing products that are out of stock or below the predefined threshold. The report should be available in CSV and PDF formats and accessible via the inventory management dashboard. The inventory manager must be able to generate this report on demand and receive an automatic daily summary via email."...</i>

Fig. 5. Resumen de la consulta genérica sobre el primer requerimiento.

Esta expresión mejorada propuesta por el modelo es luego evaluada contra la obtenida con la implementación simplificada del marco propuesto.

4.4 Evaluación de las expresiones con el marco propuesto

En esta sección se desarrolla una Prueba Conceptual para evaluar la factibilidad del enfoque. Para simplificar el ejemplo se utiliza ChatGPT como único modelo en las distintas etapas. Esta implementación del marco requiere tres modelos, a) evaluador de la expresión, b) generador de preguntas y c) generador de una nueva expresión mejorada. La decisión sobre la calidad del requerimiento la toma el analista y las respuestas a las preguntas que alimentan el último modelo fueron generadas para estas pruebas. No se utiliza un agente, ya que el analista asume ese rol coordinando los componentes.

Modelo Evaluador (a). La propuesta requiere definir previamente cómo se evaluarán los requerimientos, lo que permite identificar con claridad los problemas en las expresiones. Dado que las pruebas se realizan con un modelo generativo, la primera interacción establece y verifica que el modelo comprenda el contexto, en este caso, la guía de INCOSE (Figura 6).

User Prompt
Do you know the characteristics of well-formed requirements: "Necessary, Appropriate, Unambiguous, Complete, Singular, Feasible, Verifiable, Correct and Conforming" published in "INCOSE Guide for writing requirements "?
Model
Yes! The INCOSE (International Council on Systems Engineering) Guide for Writing Requirements defines well-formed requirements using the following characteristics: 1. Necessary – The requirement must be essential to meet stakeholder needs ...

Fig. 6. Resumen de la primera interacción con el modelo.

Luego de esta interacción inicial, se procede a evaluar el requerimiento. El prompt está compuesto por: la tarea a realizar, el texto del requerimiento y, por último, define el formato deseado para la respuesta (Figura 7).

Con la salida obtenida en la última interacción se puede tomar la decisión sugerida en el marco, acerca de si el requerimiento tiene la calidad esperada o no. En este caso, la decisión es tomada por el analista, aunque el marco deja abierta esta decisión que podría ser tomada de forma automática implementando un agente.

User Prompt		
Verify that the requirement meets these characteristics. Requirement: "The system must allow the inventory manager to generate a list of missing products." Your answer should be only a table with the columns [Feature Name, Feature Detail, Fulfilled (yes/no)]		
Model		
Feature Name	Feature Detail	Fulfilled (yes/no)
Necessary	Essential for inventory management	Yes
Appropriate	At the correct level for system requirements	Yes
Unambiguous	Clearly defines the function	No
Complete	Contains all necessary information	No
Singular	Addresses only one function	Yes
Feasible	Achievable with existing technology	Yes
Verifiable	Testable via inspection or system validation	No
Correct	Accurately describes the needed function	Yes
Conforming	Follows standard requirement guidelines	No

Fig. 7. Salida del modelo evaluador para el primer requerimiento.

Del análisis del ejemplo de la Figura 7 se concluye que el requerimiento no tiene la calidad esperada debido a que, según la respuesta del modelo, tres características deseadas en la propuesta de INCOSE no se cumplen. Estas características son “No ambiguo”, “Completo” y “Verificable”. La característica “Conforme” se desestima en todas las evaluaciones, ya que estas pruebas no incluyen información acerca de los estándares de la organización para la escritura de requerimientos.

Modelo Generador de Preguntas (b). La segunda interacción con un modelo tiene como fin obtener preguntas cuyas respuestas mejoren la expresión en las características no cumplidas, según la interacción anterior. Para este propósito se confecciona un prompt que describe la tarea a realizar y el formato de la salida esperada (Figura 8).

User Prompt		
Define questions whose answers help to comply with the missing characteristics: "unambiguous, complete and verifiable". Your answer should only be a table with the columns [Feature Name (unambiguous, complete, or verifiable), suggested questions]		
Model		

Feature Name	Suggested Questions
Unambiguous	What format should the generated list follow (e.g., PDF, CSV, on-screen display)?...
Complete	Should the system allow filtering or sorting the missing product list? ...
Verifiable	What criteria will be used to confirm the list is correctly generated?

Fig. 8. Salida resumida del modelo generador de preguntas para el primer requerimiento.

Modelo Generador de Requerimientos (c). A partir de las preguntas generadas en la última interacción, se selecciona un conjunto que cubra todos los aspectos a mejorar y se generan respuestas sintéticas con el fin de culminar la aplicación del marco. Se confecciona un prompt como entrada al último modelo, que describe la tarea a realizar e incluye el texto original del requerimiento, las preguntas y respuestas para mejorarlo y los formatos propuestos para la redacción de requerimientos (Figura 9). En la Figura 9, se observa que el modelo propone un requerimiento mejorado. Esta expresión incluye detalles extraídos por el modelo de las respuestas provistas.

User Prompt
Improve the requirement using the answers to questions. The improved requirement must be expressed according to the Proposed Format. Requirement: "The system must allow the inventory manager to generate a list of missing products." Questions and answers: Q1-What format should the generated list follow (e.g., PDF, CSV, on-screen display)? A1-The list should be available in PDF and CSV formats, as well as an on-screen display for quick review. Q2-... Proposed formats: F1- The <entity> shall <action verb> <object> <measurable outcome>. F2- When <condition clause>, the <subject clause> shall <action verb clause> <object clause> <qualifying clause>.
Model
Improved Requirement (Format F2): <i>When the inventory manager or authorized personnel request a missing products list, the system shall generate a report that includes out-of-stock items and products below a predefined threshold, providing details such as product name, current quantity, supplier, and recommended reorder date. The list shall be available in PDF and CSV formats, as well as an on-screen display, and shall support filtering by category, supplier, and stock status, with sorting by priority.</i>

Fig. 9. Salida resumida del modelo generador de preguntas para el primer requerimiento.

4.5 Evaluación

Para evaluar las expresiones obtenidas con el marco y facilitar la comparación con las obtenidas según la propuesta de la sección 4.3, se utilizó el mismo prompt que representa el modelo evaluador (a) descripto en la Figura 6. Por cada requerimiento se

evaluó la expresión original (RO), la generada con un prompt genérico (RG) y la generada con la implementación del marco (RM) (Tabla 3).

Tabla 3. Resultados de la evaluación sobre el requerimiento original (**RO**), el generado con el prompt genérico (**RG**) y el generado con la implementación del marco (**RM**).

#	Requerimiento 1			Requerimiento 2			Requerimiento 3		
Característica	RO	RG	RM	RO	RG	RM	RO	RG	RM
Necesario	Si	Si	Si	Si	Si	Si	Si	Si	Si
Apropiado	Si	Si	Si	No	Si	Si	Si	Si	Si
No ambiguo	No	No	Si	No	Si	Si	No	No	Si
Completo	No	No	Si	No	Si	Si	No	No	Si
Singular	Si	No	No	No	No	Si	Si	Si	No
Factible	Si	Si	Si	Si	Si	Si	Si	Si	Si
Verificable	No	Si	Si	No	Si	Si	Si	No	Si
Correcto	Si	Si	Si	No	Si	Si	Si	Si	Si
% Caract. Cumplidas	62.5	62.5	87.5	25	87.5	100	100	75	50
									100

Para interpretar la Tabla 3, se aclara que para el Requerimiento 2 existe una columna extra en la expresión obtenida con el marco propuesto. Esto se debe a que la propuesta detectó la no singularidad y generó dos expresiones para solucionarla.

La “calidad” de las expresiones se evalúa a partir de la puntuación devuelta por el modelo, utilizando una métrica que refleja el porcentaje de características cumplidas por cada una. Esta medida, presentada en la última fila de la tabla, muestra que el marco propuesto alcanza los mejores resultados, logrando un 100 % en tres casos (Figura 10).

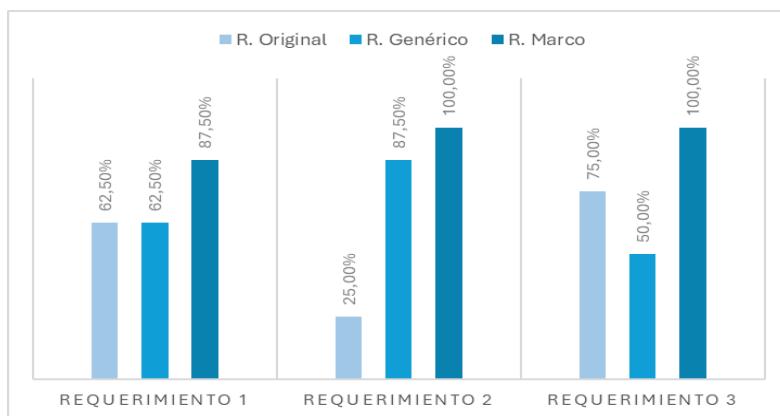


Fig. 10. Comparación, según la métrica propuesta, de la calidad de los distintos requerimientos.

5 Conclusiones y Trabajos Futuros

El framework propuesto tiene un gran potencial, sin embargo, es esencial reconocer que sólo se presenta una prueba conceptual. La implementación utilizada en este trabajo es sencilla y hace uso de sólo tres modelos representados por interacciones con ChatGPT. Sin embargo, estas pruebas preliminares sirven para recoger los primeros resultados y conclusiones, así como también guiar futuros trabajos.

La estrategia de dividir el problema en tareas simples y pequeños incrementos a la solución, aporta a los modelos el conocimiento y el contexto que se va acumulando luego de cada interacción. Esto no es menor, ya que ese contexto generado en los pasos previos suma información a la entrada de la interacción que permite una respuesta más precisa y completa. Esto puede observarse, por ejemplo, en el Requerimiento 2, donde primero se detecta la no singularidad, luego se genera una pregunta acerca de la posibilidad de dividir la expresión y, por último, ante una respuesta afirmativa a esta pregunta, el modelo genera dos expresiones.

Igualmente, es de destacar cómo esta implementación simplificada del marco pudo mejorar las expresiones hasta lograr un 96.78% en la métrica propuesta, promediando las medidas logradas para las cuatro expresiones generadas.

Además, se puede observar que las expresiones generadas por una consulta genérica no mejoran en todos los casos la expresión original, empeorándola a veces como en el requerimiento 3. Estas expresiones logran un puntaje de 66.6% con la métrica propuesta y promediando los puntajes. Puede observarse, por ejemplo, que la consulta genérica no soluciona la no singularidad e inclusive, a veces, la causa.

Por último, se destaca cómo el último modelo generador de expresiones de requerimientos puede crearlas siguiendo patrones que son anexados al prompt de generación. Son varios los posibles trabajos futuros que se desprenden de la propuesta, como ser la implementación de un prototipo de software funcional que implemente un agente para automatizar el proceso y así permitir su aplicación con datos reales de proyectos de desarrollo de software. Esto permitiría evaluar la propuesta a mayor escala y con distintas implementaciones del marco.

La variedad de modelos y artefactos posibles exige una mejor y más completa definición de la propuesta, y una evaluación sobre qué interfaces y repositorios se pueden utilizar para conectar los modelos, además de cómo implementar la vectorización de texto en la base de datos -en el caso de utilizar RAG- o cómo entrenar modelos y preparar los conjuntos de datos son algunos de los desafíos.

Referencias

- Brown, Tom & Mann, Benjamin & Ryder, Nick & Subbiah, Melanie & Kaplan, Jared & Dhariwal, Prafulla & Neelakantan, Arvind & Shyam, Pranav & Sastry, Girish & Askell, Amanda & Agarwal, Sandhini & Herbert-Voss, Ariel & Krueger, Gretchen & Henighan, Tom & Child, Rewon & Ramesh, Aditya & Ziegler, Daniel & Wu, Jeffrey & Winter, Clemens & Amodei, Dario. (2020). Language Models are Few-Shot Learners.
ChatGPT Homepage, <https://chatgpt.com/?model=auto>, last accessed 28/03/2025.
Copilot Homepage, <https://copilot.microsoft.com>, last accessed 28/03/2025.

- Devlin, Jacob & Chang, Ming-Wei & Lee, Kenton & Toutanova, Kristina. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Gemini Homepage, <https://gemini.google.com/app>, last accessed 26/03/2025.
- GtWR Homepage, https://www.incose.org/docs/default-source/working-groups/requirements-wg/gtwr/incose_rwg_gtwr_v4_040423_final_drafts.pdf?sfvrsn=5c877fc7_2 last accessed 27/03/2025.
- Klesel, M., Wittmann, H.F. Retrieval-Augmented Generation (RAG). Bus Inf Syst Eng (2025). <https://doi.org/10.1007/s12599-025-00945-3>
- Liu, Yixin & Shi, Kejian & He, Katherine & Ye, Longtian & Fabbri, Alexander & Liu, Pengfei & Radev, Dragomir & Cohan, Arman. (2024). On Learning to Summarize with Large Language Models as References. 8647-8664. 10.18653/v1/2024.naacl-long.478.
- Norheim, J. J., Rebentisch, E., Xiao, D., Draeger, L., Kerbrat, A., & de Weck, O. L. (2024). Challenges in applying large language models to requirements engineering tasks. Design Science, 10, e16. doi:10.1017/dsj.2024.8
- Nuseibeh, B., & Easterbrook, S. (2000). Requirements engineering. Proceedings of the Conference on The Future of Software Engineering. <https://doi.org/10.1145/336512.336523>.
- Porter, D., DeFranco, J. F., & Laplante, P. (2025). Requirements specification automated quality analysis: Past, present, and future. Computer, 58(1), 101–104. <https://doi.org/10.1109/MC.2024.3480629>.
- Prompt Engineering Guide. (2024). Elements of a prompt. In Prompting Guide. <https://www.promptingguide.ai/introduction/elements>, last accessed 28/03/2025.
- Raffel, Colin & Shazeer, Noam & Roberts, Adam & Lee, Katherine & Narang, Sharan & Matena, Michael & Zhou, Yanqi & Li, Wei & Liu, Peter. (2019). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer.
- Reinpold, L. M., Schieseck, M., Wagner, L. P., Gehlhoff, F., & Fay, A. (2024). Exploring LLMs for verifying technical system specifications against requirements. En 2024 IEEE 3rd Industrial Electronics Society Annual On-Line Conference (ONCON) (pp. 1–6). IEEE. <https://doi.org/10.1109/ONCON62778.2024.10931625>.
- Robinson, J., Ryting, C., & Wingate, D. (2022). Leveraging Large Language Models for Multiple Choice Question Answering. *International Conference on Learning Representations, abs/2210.12353*. <https://doi.org/10.48550/arXiv.2210.12353>.
- Sommerville, Ian (2016). Software Engineering (10th ed.). Addison-Wesley. ISBN 978-0-13-703515-1.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems (p./pp. 5998--6008).
- Volk, Martin; Fischer, Dominic P; Fischer, Lukas; Scheurer, Patricia; Ströbel, Phillip (2024). LLM-based Machine Translation and Summarization for Latin. In: Third Workshop on Language Technologies for Historical and Ancient Languages. <https://doi.org/10.5167/uzh-259369>.
- Wiegers, K., & Beatty, J. (2013). Software requirements (3^a ed.). Microsoft Press. ISBN: 978-0735679665.